TCG Mobile Abstraction Layer

Specification Version 1.0 Revision 2.03
29 April 2010

Contact: admin@trustedcomputinggroup.org

TCG PUBLISHED
Copyright © TCG 2010

TCG

Copyright © 2010 Trusted Computing Group, Incorporated.

Disclaimer, Notices and License Terms

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. Without limitation, TCG disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification and to the implementation of this specification, and TCG disclaims all liability for cost of procurement of substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, direct, indirect, or special damages, whether under contract, tort, warranty or otherwise, arising in any way out of use or reliance upon this specification or any information herein.

This document is copyrighted by Trusted Computing Group (TCG), and no license, express or implied, is granted herein other than as follows: You may not copy or reproduce the document or distribute it to others without written permission from TCG, except that you may freely do so for the purposes of (a) examining or implementing TCG specifications or (b) developing, testing, or promoting information technology standards and best practices, so long as you distribute the document with these disclaimers, notices, and license terms.

Contact the Trusted Computing Group at www.trustedcomputinggroup.org for information on specification licensing through membership agreements.

Any marks and brands contained herein are the property of their respective owners.

Table of Contents

1. Scope and Audience 6
1.1 Key words 6
1.2 Statement Type 6
1.3 References 7
2. Basic Definitions 8
2.1 Glossary 8
2.2 Representation of Information 8
2.2.1 Endness of Structures 8
2.2.2 Byte Packing 8
2.2.3 Lengths 8
2.3 Defines 9
2.3.1 Basic data types 9
2.3.2 Boolean types 9
2.3.3 Structure Tags ..... **Error! Bookmark not defined.**
2.3.4 MTM and Reference Architecture structures and datatypes 9
2.4 Overview 10
3. Introduction 11
4. Components and Operation of the TBB 12
4.1 Engine Model Overview 12
4.1.1 List of TBB Components 14
4.2 TBB Operation 16
4.2.1 Boot-time Operation of the TBB 17
4.2.2 Run-time Operation of the TBB 17
4.2.3 Installations and Updates performed using the TBB 18
4.3 Example Platform instantiation using TBB 19
5. Ordinal-Level Interface to the TBB 21
5.1 MTM Ordinals Exposed by the TBB 22
5.1.1 Table of Ordinals 22
5.1.2 Trust Requirements for Use of the Ordinals 26
5.2 Additional Ordinals Exposed by the TBB 31
5.3 Communication with Secure RIM Conversion Agent 33

5.3.1 TBB_AddExternalRIMData 34
5.3.2 TBB_RemoveExternalRIMData 38
5.3.3 TBB_InternalizeRIMCertificate 39
5.3.4 TBB_GenerateInternalRIMCertificate 41
5.3.5 TBB_RefreshInternalRIMCertificate 44
5.3.6 TBB_RemoveInternalRIMCertificate 45
5.4 Look-up from a Secure RIM Database 47
5.4.1 TBB_FindInternalRIMCertificate 47
5.4.2 TBB_EnumInternalRIMCertificates 49
5.5 Communication with PRMVA/Watchdog Timer 51
5.5.1 TBB_ConfigureWDTMeasurements 54
5.5.2 TBB_SeedASC 56
5.5.3 TBB_IncrementASC 57
5.6 Additional (Virtual) Monotonic Counters and NV Storage 59
5.6.1 TBB_ProtectVirtualCountersAndStorage 60
5.7 Use restricted commands (e.g. MakeIdentity/ActivateIdentity) 62
6. TBB Interface API 64
6.1 TDDL Interface 65
6.1.1 Tddli_Open 65
6.1.2 Tddli_Close 65
6.1.3 Tddli_Cancel 66
6.1.4 Tddli_GetCapability 66
6.1.5 Tddli_SetCapability 68
6.1.6 Tddli_GetStatus 68
6.1.7 Tddli_TransmitData 69
6.1.8 Tddli_PowerManagement 70
6.1.9 Tddli_PowerManagementControl 72
6.2 TBB Interface Implementation 74
6.2.1 Unfiltered TDDLI 74
6.2.2 Filtered TDDLI 75
6.3 Recommended Interface API 77
6.3.1 Use of Java in Mobile Platforms 77

7. Trusted Services API 78
7.1 Implementation at TCSI Level of TSS 79
7.1.1 Classification of TCSI 79
7.1.2 Additional APIs related to TBB 82
7.2 Implementation at TSPI Level of TSS 99
7.2.1 Classification of TSPI 99
7.2.2 Additional APIs at TSPI Level 102
7.2.3 Interface Between Engines 116

1. Scope and Audience

The TCG specifications define a Trusted Platform Module (TPM) and its use in various computing platforms. The "TCG Mobile Trusted Module Specification" [5] is a specification that defines the necessary interface for implementing Mobile Trusted Modules. The "TCG Mobile Reference Architecture Specification" [9] defines ways of instantiating and using Mobile Trusted Modules in a trusted mobile platform. This specification defines an abstraction layer to the trusted components of a mobile platform as described in [5] and [9].

This document is an industry specification that enables the building of trust in mobile phones using a standardized approach.

1.1 Key words

The key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" in the chapters 2-7 normative statements are to be interpreted as described in [RFC-2119].

1.2 Statement Type

Please note a very important distinction between different sections of text throughout this document. You will encounter two distinctive kinds of text: *informative comment* and *normative statements*. Because most of the text in this specification will be of the kind *normative statements*, the authors have informally defined it as the default and, as such, have specifically called out text of the kind *informative comment*. They have done this by flagging the beginning and end of each *informative comment* and highlighting its text in gray. This means that unless text is specifically marked as of the kind *informative comment*, you can consider it of the kind *normative statements*.

For example:

Start of informative comment:

This is the first paragraph of 1-n paragraphs containing text of the kind ***informative comment*** ...

This is the second paragraph of text of the kind ***informative comment*** ...

This is the nth paragraph of text of the kind ***informative comment*** ...

To understand the TPM specification the user must read the specification. (This use of MUST does not require any action).

End of informative comment.

This is the first paragraph of one or more paragraphs (and/or sections) containing the text of the kind *normative statements* ...

To understand the TPM specification the user MUST read the specification. (This use of MUST indicates a keyword usage and requires an action).

1.3 References

[1] Trusted Computing Group, *TPM Main Part 1 Design Principles,* Specification Version 1.2 Revision 103, July 2007

[2] Trusted Computing Group, *TPM Main Part 2 TPM Structures,* Specification Version 1.2 Revision 103, July 2007

[3] Trusted Computing Group, *TPM Main Part 3 Commands,* Specification Version 1.2 Revision 103, July 2007

[4] Trusted Computing Group, *Mobile Phone Work Group Use Case Scenarios,* Specification Version 2.7, 2005.

[5] Trusted Computing Group, *TCG Mobile Trusted Module Specification*, Version 1.0 Revision 6, June 2008

[6] Trusted Computing Group, *TCG Credential Profiles*, Specification Version 1.1, Revision 1.014, 21 May 2007, For TPM Family 1.2; Level 2

[7] Trusted Computing Group,*TCG Glossary of Technical Terms,* https://www.trustedcomputinggroup.org/groups/glossary/

[8] Trusted Computing Group, *TCG Specification Architecture Overview*, Revision 1.4, August 2007

[9] Trusted Computing Group, TCG Mobile Reference Architecture Version 1.0 Revision 5, June 2008

[10] Trusted Computing Group, TCG Software Stack (TSS) Specification Version 1.10 Golden, August 2003

2. Basic Definitions

Start of informative comment:

The following structures and formats describe the interoperable areas of the specification. There is no requirement that internal storage or memory representations of data must follow these structures. These requirements are in place only during the movement of data from a Mobile Trusted Module (MTM) to some other entity.

End of informative comment.

2.1 Glossary

Other TCG technical terms are as defined in the TCG Glossary [7] and in existing TCG Mobile Specs [5], [9]. The following descriptions does not imply any Trusted Computing Group endorsement of these definitions.

Abbrevation	Description
Trusted Building Block	From http://dblp.l3s.de/d2r/page/publications/conf/csreaSAM/Smith04 : The TCG trusted building block is the minimum hardware/software needed to perform measurement, storage and reporting. It is trusted to operate correctly. Reporting platform configuration allows others (verifiers) to vet configuration state according to acceptable values.
Trusted Computing Base	Security architectures are best done using a layered and simple approach separating the components that require trust from the components that don't. An architecture where lines can be drawn around and between the trusted and untrusted modules allows for easier maintenance and validation of the security properties of the system. This line is historically called the Trusted Computing Base or TCB. A TCB must be able to defend itself since maintaining the TCB is essential for security policy to be implemented successfully. Reference: http://www.fas.org/irp/nsa/rainbow/tg003.htm

2.2 Representation of Information

2.2.1 Endness of Structures

Each structure MUST use big endian bit ordering, which follows the Internet standard and requires that the low-order bit appear to the far right of a word, buffer, wire format, or other area and the high-order bit appear to the far left.

2.2.2 Byte Packing

All structures MUST be packed on a byte boundary.

2.2.3 Lengths

The “Byte” is the unit of length when the length of a parameter is specified.

2.3 Defines

Start of informative comment:

These definitions are in use to make a consistent use of values throughout the structure specifications. The types in sections 2.3.1, 2.3.2 and 2.3.3 are reproduced here for the reader's convenience. This document fully re-uses the type definitions from [2] as reproduced in Sections 2.3.1 and 2.3.2.

End of informative comment.

2.3.1 Basic data types

Typedef	Name	Description
unsigned char	BYTE	Basic byte used to transmit all character fields.
unsigned char	BOOL	TRUE/FALSE field. TRUE = 0x01, FALSE = 0x00
unsigned short	UINT16	16-bit field. The definition in different architectures may need to specify 16 bits instead of the short definition
unsigned long	UINT32	32-bit field. The definition in different architectures may need to specify 32 bits instead of the long definition

2.3.2 Boolean types

Name	Value	Description
TRUE	0x01	Assertion
FALSE	0x00	Contradiction

2.3.3 MTM and Reference Architecture structures and datatypes

All type and structure definitions used in this specification are defined in [2], [5] and [9]

2.4 Overview

This specification defines an abstraction layer to the trusted components of a mobile platform. This specification does not define how the trusted components must be implemented, but does define their functional properties in terms of an interface exposed by the trusted components towards the normal (untrusted) components of the platform. Collectively, the trusted components are referred to as the Mobile Trusted Building Block (TBB).

The scope of the specification is:

- The definition of the mobile TBB for any given engine in a mobile platform, in terms of the components defined in the MTM [5] and Reference Architecture [9].
- A description of the operations of the mobile TBB components and how they interact with the normal (untrusted) components
- The definition of a low level (ordinal, channel) interface towards the mobile TBB
- A profile for a higher level interface towards the TBB, based on layers of the TSS [10]

3. Introduction

Start of informative comment:

This specification is intended to provide an abstraction layer to the MTM [5] and other trusted components of a mobile platform, as identified in the TCG Mobile Reference architecture[9]. The specification has the following objectives

1. To define a TBB (full set of RoTs, protected capabilities) in terms of the interface used to access the TBB, and the assets protected by the TBB.

   - To ensure it is the same interface regardless of physical implementation
   - To provide a consistent low level interface: essentially ordinal level.

   The basic goal here is to allow adoption of TCG specs by multiple physical implementations, while also allowing consistent security statements to be made about the TBB implementation. Ultimately, the interface could be used to define a Protection Profile (and hence conformance criteria) for a full TBB.

2. To Facilitate MTM testing (via test vectors, harness etc.)

   - This requires a low-level interface, just a pipe from the test harness to MTM, used to carry ordinals, like the TDDLI in the TSS.

   The basic goal here is to ensure that MTM implementations are functionally consistent (and interoperable with a test suite) across a range of platforms. The abstraction layer will allow software (such as a test harness) to be developed for use across a range of mobile platforms, without having to be customized heavily based on the physical implementation of an MTM in each platform.

3. To Provide higher level interface/API.

   - For instance to facilitate software design without use of MTM ordinals, and maybe abstract the MTM away completely.
   - To provide a logic for one engine to export trusted services to another
   - Like the TSPI in the TSS, or more abstract still.

   The basic goal here is to speed software development, and also facilitate consistency of software development between PC implementations (using the TSS) and mobile implementations (using a mobile profile of the TSS).

It will be seen that the objectives require **two** levels of abstraction, one much more low level, and that the higher level of abstraction depends on the lower level. The specification is therefore structured as follows.

Chapter 4 provides an overview of a Trusted Mobile Engine, extending the model described in [9]. The role of the TBB as a set of trusted resources is defined, and the place for an abstraction layer as the interface between normal services and trusted resources is shown. The chapter provides a list of mandatory and optional TBB components, and describes the operation of the TBB during the engine's lifecycle.

Chapter 5 defines the ordinal level interface to the TBB, both in terms of MTM ordinals, and additional TBB ordinals which can be used to support the lifecycle processes described in Chapter 4. Chapters 4 and 5 together aim to fulfill objective 1 listed above.

Chapter 6 defines a TBB Interface API in terms of a "pipe" used to carry ordinals, and shows how it extends the TSS TDDLI. Requirements for two versions of the "pipe" – one for restricted ordinals, which can only be called by trusted services, one for unrestricted ordinals – are discussed. It is noted that this API can be provided as a Java API for use in a Java TSS or test-suite. Chapter 6 aims to fulfill objective 2 listed above.

Chapter 7 then provides a cut-down profile of the existing TSS [10] for use on mobile platforms. Profiles are given for the Core Services and Service Provider interface layers. Chapter 7 aims to fulfill objective 3 above.

End of informative comment.

4. Components and Operation of the TBB

4.1 Engine Model Overview

Start of informative comment:

The interfaces defined in this abstraction layer specification are based on the Generic Engine description as provided in the Mobile Phone Reference Architecture [9].

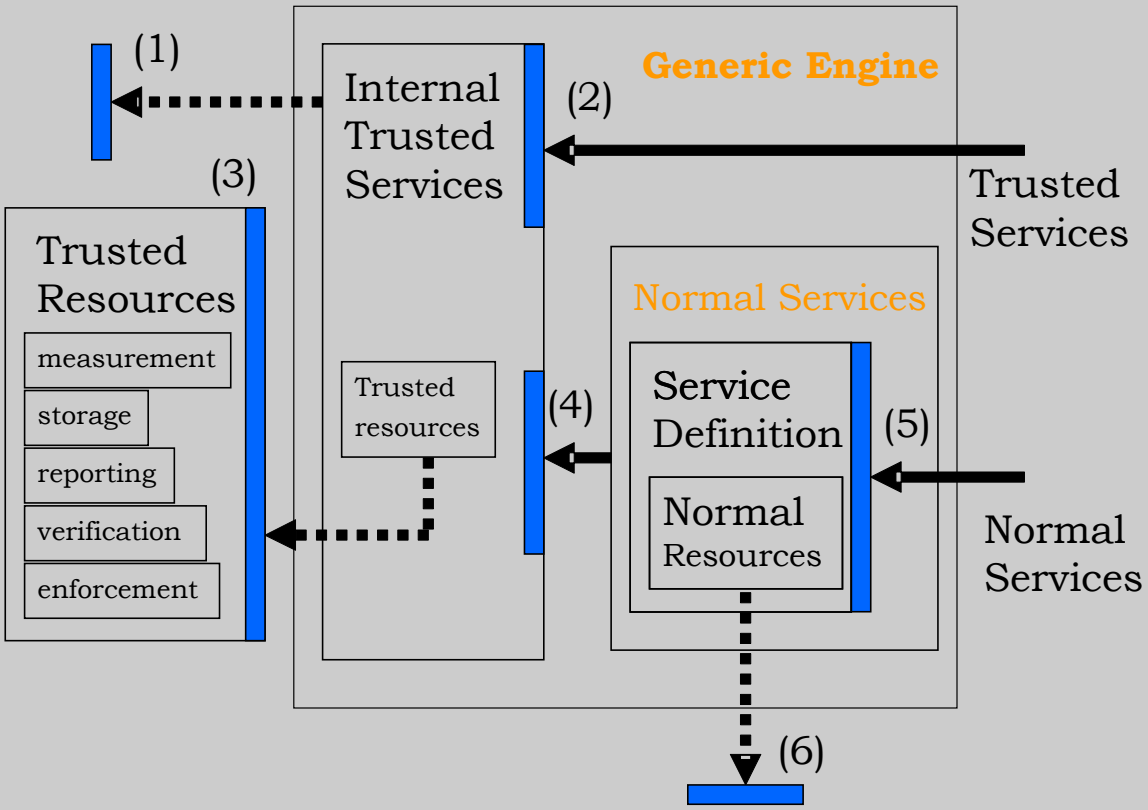


Figure 1. Generic Engine

The normative requirements in the next paragraph repeat requirements from the Mobile Reference Architecture specification, and serve to define the interfaces 2, 3 and 4.

End of informative comment

Each engine MUST contain Internal Trusted Services. Internal Trusted Services MUST measure (interface #4) Normal Services and SHOULD provide other services (such as Protected Storage, Monotonic Counters, Delegation, etc.) to the Normal Services. The Internal Trusted Services MUST be able to report (interface #2) measurements (signed by AIKs) to external entities, and SHOULD use those measurements internally (for sealed data). Internal Trusted Services MUST have internal access (interface #3) to a set of Trusted Resources.

Within this specification, the full set of Trusted Resources for an engine is referred to as the engine's Trusted Building Block (TBB). Interface #4 is defined as a "Trusted Services API" and interface #3 is defined as a "TBB Interface API".

As discussed in the Mobile Reference Architecture, the Trusted Resources can be either dedicated or allocated and MAY be provided to the engine by another engine (interface #1). For example, the Trusted Resources for all engines MAY be provided by the Device Manufacturer's engine. Where one engine exports Trusted Services to another engine, to be used as the dependent engine's Trusted Resources, then the

exporting engine SHOULD use Interface #2. A dependent engine MAY then use interface #2 of the Device Manufacturer's engine to import its own interface #1. The dependent engine SHOULD then use the imported interface #1 to provide either its own interface #3 (TBB Interface API) or interface #4 (Trusted Services API).

Start of informative comment:

A more detailed expansion of the Trusted Resources and the Normal Services and the interfaces between them is now shown in Figure 2 below.

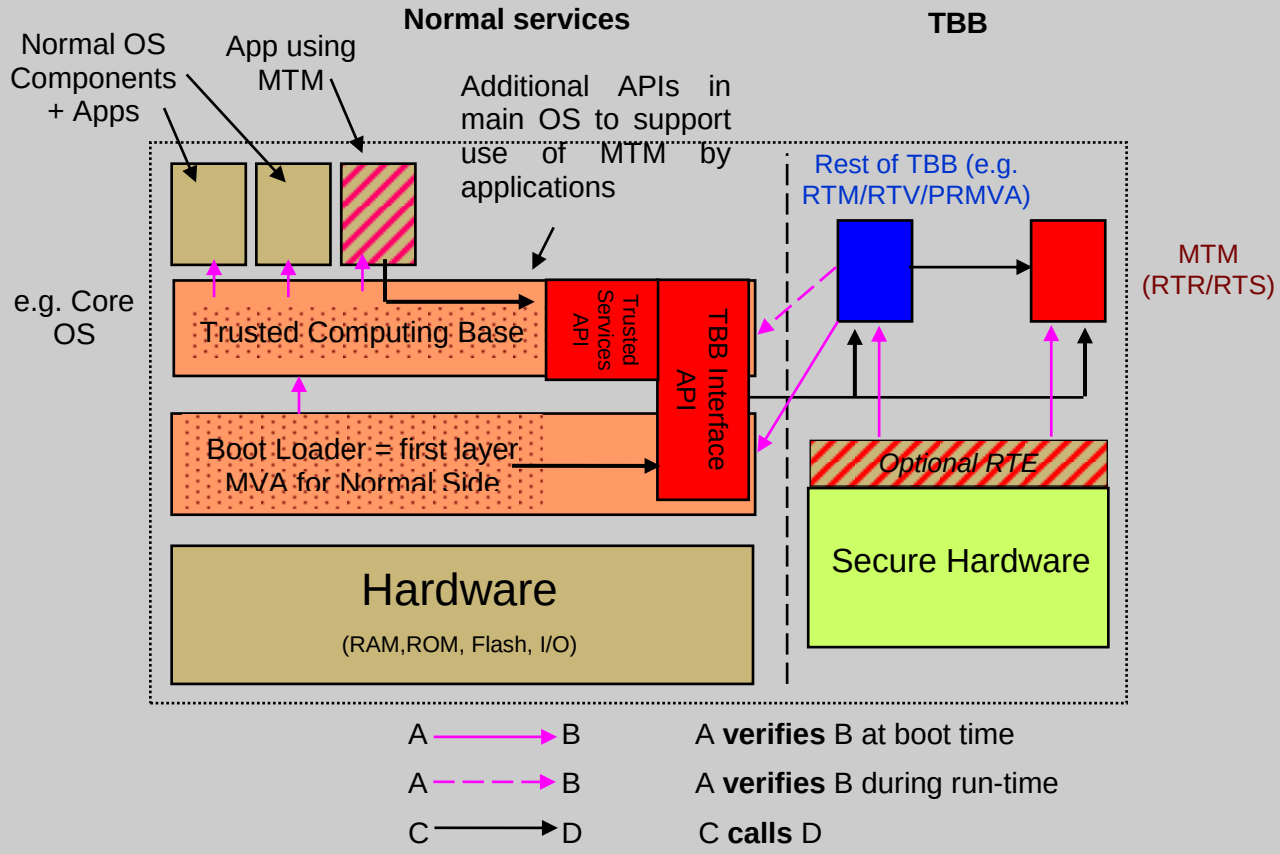


Figure 2. Interfaces from Normal Services to the TBB

End of informative comment

During engine run-time, a core set of Trusted Services is needed to secure the operation of other Trusted Services and to ensure that Normal Services run with the intended privileges and degree of isolation. This core set of Trusted Services is conventionally referred to as a Trusted Computing Base (TCB).

For security reasons, the TCB SHOULD have a minimal code-base, and SHOULD operate in a way which enables its integrity to be readily verified at boot-time and during run-time (e.g. the TCB has a static run-time image). The engine MAY support an explicit virtualization model, in which case the TCB SHOULD be implemented as a mini-OS (hypervisor) on top of which one or more full Operating Systems are run to provide the engine's Normal Services. In that case, additional dependent engines MAY also be implemented with their own Operating Systems running on top of the base engine's TCB.

Alternatively the engine MAY be implemented using a single conventional operating system, in which case the TCB SHOULD be implemented as a minimal OS kernel. Additional dependent engines MAY then be

implemented as groups of related applications running within the single OS, with each group having defined API privileges and its own security policies governing which applications can be installed in each engine and use those API privileges.

In the situation where engines operate within a Secure Execution Environment the environment MAY support the ability to securely swap both applications and data in and out of the secure environment.

4.1.1 List of TBB Components

Start of informative comment:

Every engine's TBB must contain the Roots of Trust for that engine, and may contain some additional components. The required components depend upon which RoTs are supported by that engine, in particular whether the engine is a secure-boot engine with an RTV. Engines with an RTV are also required to support reactive capabilities to respond to an integrity failure, and at least one continuing MVA which can detect such an integrity failure.

End of informative comment

For all engines, the following three components are **Mandatory** Trusted Resources (TBB components):

1. A Root-of-Trust-for-Storage (RTS), which provides PCRs and Protected Storage for an engine. The RTS stores the measurements made by the Root-of-Trust-for-Measurement (or its subsequent measurement agents), cryptographic keys, and security sensitive data.

2. A Root-of-Trust-for-Reporting (RTR), which reports the measurements stored by the Root-of-Trust-for-Storage by signing them with cryptographic signature keys stored by the Root-of-Trust-For-Storage.

The RTS and RTR together form the **Mobile Trusted Module**, as described in [5].

3. A Root-of-Trust-for-Measurement (RTM), which performs the measurement functionality as defined in the TPM Main Specification.

In addition, the following three components are mandatory **if** the engine has secure boot:

1. A Root-of-Trust-for-Verification (RTV), which checks measurements against reference integrity metrics before they are extended into the RTS. The RTV may verify the measurements of the current state of other engines. The RTV can reliably verify a measured integrity metric against a RIM, and extend the integrity metric into a Platform Configuration Register (PCR)

The RTM and RTV can be combined in a single piece of code, or can be combined with the MTM as a single component as discussed in [9].

2. The TCG_Reactive capabilities, which MUST enforce a designated security response on discovery of an integrity failure in engine mandatory functions, as discussed in [9]

3. A Primary Run-time Measurement and Verification Agent (PRMVA) which carries on running as a mandatory function after OS start-up, and MUST run within protected capabilities.

This continuing MVA is **not** required to be the RTV, but can be thought of as a run-time continuation of the RTV, as discussed in [9].

Start of informative comment:

A special trusted dedicated entity may be required to build any of the RTM, RTV and RTS using allocated resources. That entity is the RTE. IF the RTE fails to build the allocated Roots-of-Trust in its engine THEN that engine is in a state that is outside the scope of this specification; an attack or failure has occurred that is beyond the capabilities of the mechanisms implicit in this specification.

An RTE is not required if the Roots of Trust of the Engine are provided as dedicated resources, or the Engine starts execution fully built (e.g. built by the Device Manufacturer's engine).

End of informative comment

TBB components composed of dedicated resources MUST perform a self-test before starting normal operation, and MUST shut-down if the test fails. If any of the Roots of Trust are to be built from allocated resources, they MUST be built by trusted resources i.e. by the RTE. If any other TBB components are built from allocated resources then they MUST be built by the Roots of Trust.

The **Optional** Trusted Resources (TBB components) for an engine include:

1. A Root-of-Trust-for-Enforcement (RTE), which is responsible for building all the Roots-of-Trust in its own engine which are based on allocated resources. The RTE is described in more detail in [9].

2. A Watchdog Timer, which is a particular functionally-defined implementation of a PRMVA and TCG_Reactive capabilities, as discussed in [9].

The following additional **Optional** components can also be regarded as delegates of the remote owner:

3. A secure RIM Conversion Agent, which holds the verificationAuth data needed to create internal RIM certificates (see [5]) and is used to convert external RIM Certificates into internal RIM Certificates.

4. Additional "virtual" Monotonic Counters, and NV Storage locations, whose values MUST be integrity protected, and SHOULD be protected using a signature mechanism which binds them to the latest value of the underlying MTM monotonic counter counterStorageProtect.

5. Additional Optional Commands (like TPM_MakeIdentity and TPM_ActivateIdentity) that are authorized by verificationAuth.

4.2 TBB Operation

Start of informative comment:

Figure 3 illustrates the interfaces to the TBB during measurement and verification.

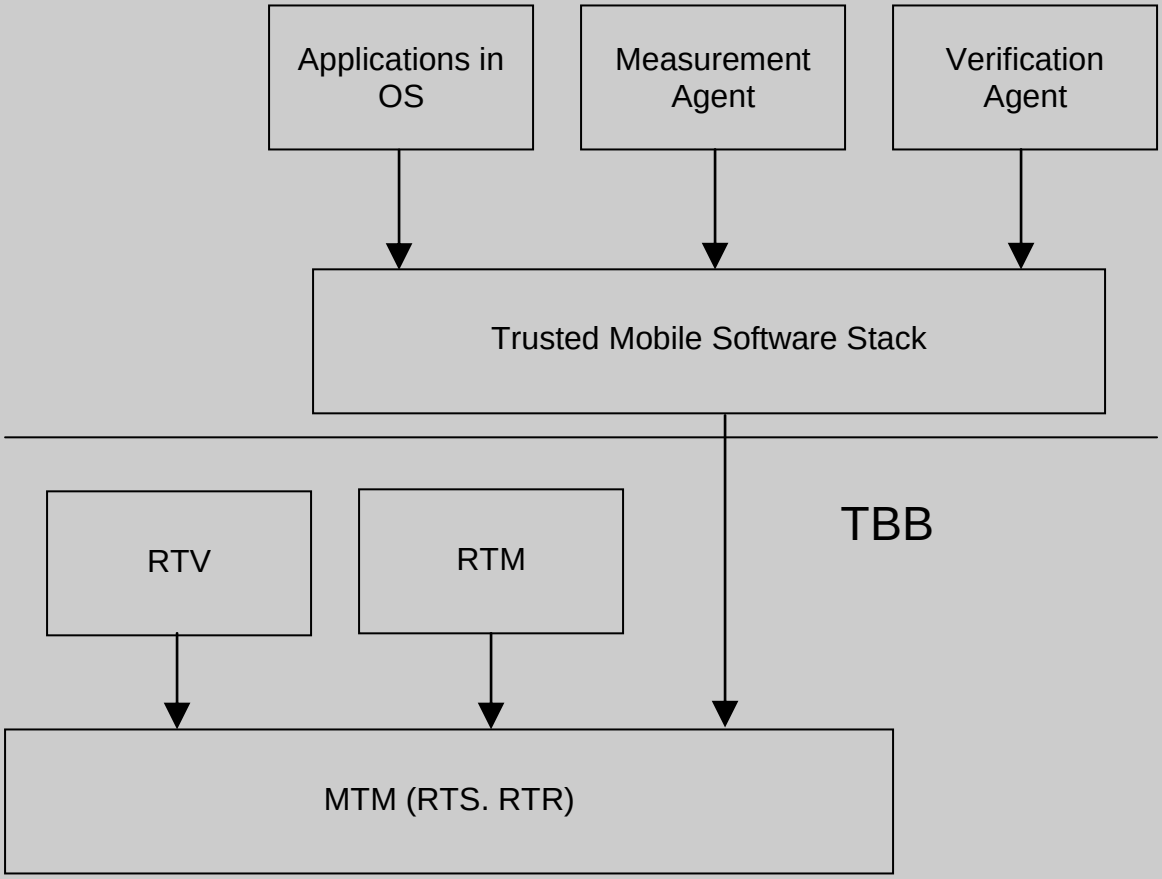


Figure 3. Interfaces

In Figure 3, the bottom is the Mobile Trusted Module [5]. On top there is a point of entry called the Trusted Mobile Software Stack (TMSS), which is defined in this specification as the conjunction of the TBB Interface API (Section 6) and Trusted Services API (Section 7),

On top of this the 'users' of the trusted services are shown. These are the OS Applications, Measurement Agents, and Verification Agents. Note, that all of those agents might actually be part of the OS, and there will be interactions between the OS and these agents, which are not depicted above. It is also important to note that in the early stages of the boot process, entities like an RTV and RTM can interact directly with the MTM without going through a TMSS.

End of informative comment.

4.2.1 Boot-time Operation of the TBB

The following steps describe the TBB operations for a secure-boot Engine.

1. If the RTE exists, then it MUST run first and MUST build a Secure Execution Environment for the Allocated Roots of Trust. This environment SHOULD contain at least two secure applications – a combined RTM/RTV application (which MAY also act later as the PRMVA), and a combined RTR/RTS application (the MTM).
2. The RTM and RTV SHALL start up the MTM, cause it to perform self-tests, load any necessary verification root key, set verified PCRs (if not pre-set), make initial measurements of the HW and the Roots of Trust, and perform verified extends into the MTM.
3. The RTM and RTV MAY measure, perform verified extends and initialize other required and optional components as required, following the guidelines laid out within the TCG Mobile Reference Architecture [9].
4. The RTM and RTV SHALL measure a Boot Loader for the Engine's Main OS, perform a verified extend of the Boot Loader's image into the MTM, and if successful, start the Boot Loader. The RTM and RTV MUST now stop running, but their code MAY later be re-used for the PRMVA.
5. The RTE MAY build the MTM into a state equivalent to that reached after the completion of step 3. In that case, the RTE MUST also perform the RTM/RTV functions, and so does not need to build a separate RTM/RTV. (But the RTE MUST still build a PRMVA.)

Start of informative comment:

The above operations are all entirely within the TBB. The option of building an MTM into a state "as if" steps 2 and 4 had been run - and not building a separate RTM/RTV - might for instance be used as an optimization during pristine boot.

The boot process then continues involving components outside the TBB. It is assumed that these components will perform the tasks described below, and will communicate with the TBB using part of the abstraction layer e.g. at least the TBB Interface API.

End of informative comment

6. The Boot Loader is assumed to measure (at least) the TCB of the Main OS and to perform a verified extend of the measurement into the MTM. An interface to the MTM MUST already be available to allow the Boot Loader to perform either an extend or verified extend and determine whether the result was successful. At least an ordinal-level interface MUST be available. If successful, it is assumed that the Boot Loader will start (at least) the TCB and then stop running.
7. The TCB is assumed to then measure (and maybe verify) other components which need to be launched to complete boot. An interface to the MTM MUST already be available to allow the TCB to perform either extends or verified extends, and determine whether the result was successful. At least, the TBB Interface API SHOULD be available.

4.2.2 Run-time Operation of the TBB

The following steps describe the intended use of the TBB during the Engine's run-time:

1. Components of the Main OS are assumed to measure (and optionally verify) any critical applications at launch. The Trusted Services API SHOULD be available, so that these components can use it to perform extends/verified extends to the MTM.
2. Components running within or on top of the Main OS are assumed to also support run-time integrity services (spot checks/alarm conditions/AV/Intrusion Detection etc.) to detect attacks on OS or on critical applications. The Trusted Services API SHOULD be available, e.g. to allow these components

to record new run-time measurements, retrieve earlier measurements, compare old measurements with new measurements.

3. The Trusted Services API SHOULD be available for use by other applications running within or on top of the Main OS e.g. to allow these applications to use the API for creating and using keys, sealing, unsealing, reading and quoting PCRs.
4. Occasionally, on a periodic or event-triggered basis, the PRMVA SHALL run in the Secure Execution Environment. The PRMVA SHOULD at least check the status of the TCB (e.g. re-measure the Main OS kernel including the SRMVA, and/or act as "Watchdog Timer"). The PRMVA SHOULD disable interrupts and debugging services while performing these checks. After verifying its integrity the PRMVA MAY request services from the TCB including requesting that the SRMVA verify the integrity of applications.

4.2.3 Installations and Updates performed using the TBB

The following steps describe the intended use of the TBB when new applications are installed which require protecting by RIMs. Where this involves the creation of a new internal RIM, then a "RIM Conversion Agent" MUST be used, and it SHOULD be split between a first component ("Normal") which runs outside the TBB, and a second component ("Secure") which runs within the TBB.

1. The first component is assumed to handle integration with a Device Management agent, verify any external signature and certificates for the new software, and unpack the new software.
2. The first component is assumed to then call the second component to measure the image of the unpacked software. The first component is also assumed to pass the second component a suitable external RIM to be converted into an internal RIM.
3. Once called, the second component MUST check that the engine state is acceptable to create a new internal RIM (e.g. PCR values allow unseal of the necessary verificationAuth data). The second component SHALL then measure the unpacked image, and – whenever it can be directly compared against the external RIM – verify that the unpacked image is as expected. The second component SHALL then use the MTM to sign a new Internal RIM; thus securely recording the application image as it was at install.
4. The first component is assumed to then register the installed software, and mark the software as being protected by the new RIM.

If applications already protected by RIMs are modified or removed entirely, or changes to the boot sequence are needed, the RIM Conversion Agent MUST be again called. The first component is assumed to call the second component to re-sign the existing internal RIMs. Once called, the second component SHALL attempt to re-sign all internal RIM Certs that remain valid. Once a complete new set of internal RIM Certs has been successfully created by the re-signing process, the second component SHALL increment counterRIMProtect to invalidate the old internal RIM Certs.

4.3 Example Platform instantiation using TBB

Start of informative comment:

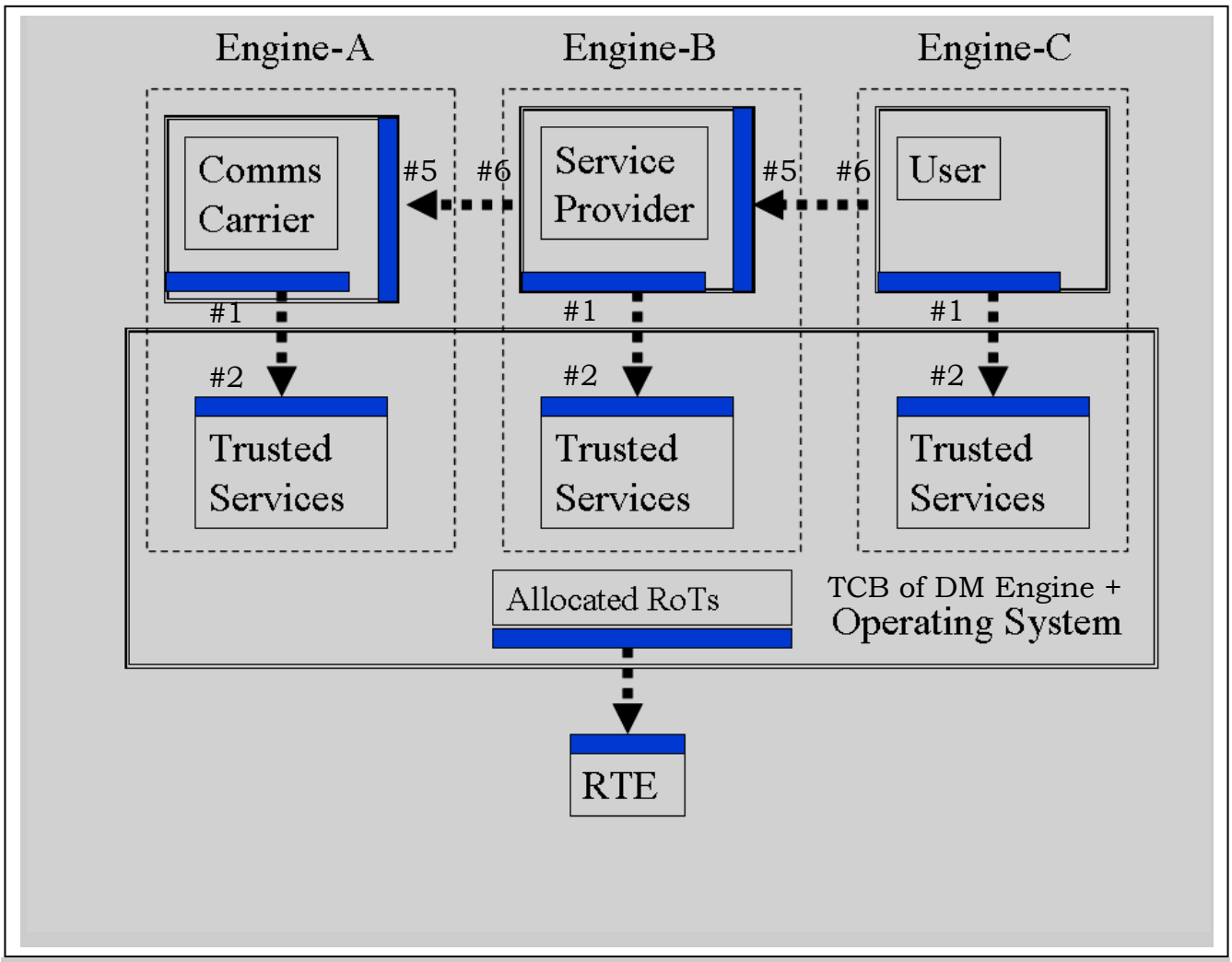


Figure 4. Example TBB Instantiation

Figure 4 illustrates a platform composed of a DM's engine and trusted engines A, B and C. In this example, only the Device Manufacturer's engine has access to basic platform resources, which comprise a computing engine with a user interface, debug connector, a radio transmitter and receiver, Random Number Generator, and SIM interface. The computing engine has a protected execution environment as well as a less privileged execution environment. The protected execution environment is used for critical security functions, in this case the Roots-of-Trust and other TBB components including a Watchdog Timer. The protected execution environment runs in a state disconnected from external devices such as debuggers.

The protected environment boots from a dedicated RTE. On boot, the RTE builds an allocated RTM, RTS, RTR and RTV in the protected environment, and then starts a Boot Loader which will load the Operating System kernel (TCB) in the less privileged environment, as well as starting a Watchdog Timer in the protected environment to provide ongoing integrity measurements of the OS kernel. The OS kernel then builds a full OS in the less privileged DM environment.

The less privileged environment accesses the protected environment exclusively through the defined TBB Interface API, and cannot subvert the protected environment. Both environments have access to volatile storage. The protected environment has access to write-once non-volatile storage in the form of ROM but has no access to protected write-many non-volatile storage. The protected environment therefore always boots into the same fixed state when power is completely removed and then reapplied. It then loads in encrypted/integrity protected state from normal NV storage.

In Figure 4, the OS of the DM's Engine provides normal OS services and also provides isolated execution environments that instantiate and isolate other trusted engines. The OS provides software compartments that instantiate trusted engines A, B and C, which are isolated sets of applications and trusted services. The trusted engines are built according to specifications provided by the Device Manufacturer, designed to satisfy the requirements of the other platform stakeholders.

Applications (i.e. measured services) executing in different trusted engines are therefore isolated from each other by the OS, are measured and reported via the OS, and receive TCG trusted services (such as Protected Storage functions and time stamping functions) from the OS. The OS also provides AIKs for each of the other trusted engines.

All the trusted services consumed by engines A, B and C are provided as a service by the DM's engine (the OS). The trusted engines A, B, and C execute applications on behalf of a Communications Carrier, Service Provider, and User. The Device Manufacturer's engine provides radio access only to the Communications Carrier. The Communications Carrier engine provides network access for the Service Provider engine, and the Service Provider engine provides an address book service for the User. The User engine stores the User's data. These services are communicated via the OS, which ensures that services come from the correct source and go to the correct destination. Each application is provided by the relevant stakeholder and stored in unprotected NV storage in the platform during manufacture.

The Trusted Services API implements the mandatory functionality (and conditional mandatory functionality because of the RTV) defined in Section 7.

End of informative comment.

5. Ordinal-Level Interface to the TBB

Start of informative comment:

The MTM is defined functionally via an ordinal-level interface that can be called by external entities. The Mobile TBB is defined in the same way. The Mobile TBB exposes a set of ordinals to the engine's "Normal Services". The ordinals exposed to different services depend on the privilege level of the service concerned, in particular, whether the service is controlled and approved by the engine's stakeholder. Some of these normal services are regarded as "privileged" in that they run in a way which the stakeholder controls and approves. Typically, these services will all be protected by RIMs, and the RIMs will be verified during engine boot and during run-time. Other services are considered "unprivileged".

End of informative comment.

The Mobile TBB is defined functionally via an ordinal-level interface, which SHALL be available to be called by external services. The Mobile TBB SHALL expose a defined set of ordinals to the engine's Normal Services. The ordinals exposed to different services SHALL depend on the privilege level of the service concerned, in particular, whether the calling service is trusted by the engine's stakeholder.

The Mobile TBB SHALL expose the MTM ordinals listed in Section 5.1 as part of its ordinal interface and MAY expose additional ordinals as listed in Section 5.2 dependent on engine functionality. For example:

- Where the engine has secure boot, the Mobile TBB SHOULD expose the ordinals listed in Section 5.3.

- Where the engine supports a Watchdog Timer, the Mobile TBB SHOULD expose the ordinals in Section 5.5.

- Where the engine's MTM has a very limited number of monotonic counters or limited NV storage, the Mobile TBB SHOULD expose the ordinals listed in Section 5.6

- Where the engine has a remote stakeholder but with local delegation of owner-authorized commands, the Mobile TBB SHOULD expose the ordinals listed in Section 5.7.

5.1 MTM Ordinals Exposed by the TBB

5.1.1 Table of Ordinals

Start of informative comment:

The following table lists all TPM and MTM ordinals and indicates whether the mobile TBB is required to expose that ordinal to services outside the TBB. The table contents depend on whether the engine supports a MRTM or an MLTM. The numbers of required ordinals are totaled at the top of the table.

End of informative comment.

Ordinals that are shown in normal text are REQUIRED to be supported by either an MRTM or an MLTM or both. Ordinals that are shown in italic text and grayed out are **not** REQUIRED to be supported by either an MRTM or an MTLM. Note that TPM_Init is not included as it does not have a defined ordinal number.

Where a cell contains a "1", then the corresponding MTM MUST support the ordinal, and the ordinal MUST be exposed by the TBB for an engine with that MTM. A "0" in a cell means that the corresponding MTM MUST **not** support the ordinal, and hence the ordinal MUST **not** be exposed by the TBB for an engine with that MTM. Thus, an ordinal shown in italic text will have the corresponding cells either blank to indicate that this is an optional ordinal for an MRTM or MLTM, or it will have a "0" to indicate that it **not** to be supported, even optionally, for an MRTM or MLTM. For example, TPM_SetOperatorAuth below is in italic text and has a '0' in the MRTM column to indicate it must **not** be implemented in an MRTM, but the MLTM column is blank to indicate that it may be implemented in an MLTM.

The text "Within TBB Only" means that the corresponding MTM MUST support the ordinal, but the TBB is **not** REQUIRED to expose that ordinal. The TBB MAY expose these ordinals for use in special modes e.g. during manufacture, or for MTM compliance purposes. In normal operations, **if** the ordinal is exposed, **and** it is called by an external process, **then** it SHALL have no effect. Either the ordinal MUST already have been called within the TBB during the initial boot sequence, or it MAY be set to have no effect anyway post manufacture (for instance in the case of MTM_LoadVerificationRootKeyDisable and MTM_SetVerifiedPCRSelection).

Total Required	**30**	**47**	**25**
Ordinal Name	**Required for MRTM**	**Required for MLTM**	**Required in Both**
MTM_InstallRIM	1		
MTM_LoadVerificationKey	1		
MTM_LoadVerificationRootKeyDisable	Within TBB Only		
MTM_VerifyRIMCert	1		
MTM_VerifyRIMCertAndExtend	1		
MTM_IncrementBootstrapCounter	1		
MTM_SetVerifiedPCRSelection	Within TBB Only		
TPM_Startup	Within TBB Only	Within TBB Only	Within TBB Only
TPM_SaveState			
TPM_SelfTestFull	1	1	1
TPM_ContinueSelfTest	Within TBB Only	Within TBB Only	Within TBB Only
TPM_GetTestResult	1	1	1

TPM_SetOwnerInstall	0	1	
TPM_OwnerSet	0	1	
TPM_PhysicalEnable	0	1	
TPM_PhysicalDisable	0	1	
TPM_PhysicalSetDeactivated	0	1	
TPM_SetTempDeactivated	0	1	
TPM_SetOperatorAuth	*0*		
TPM_TakeOwnership		1	
TPM_OwnerClear	0	1	
TPM_ForceClear	0	1	
TPM_DisableOwnerClear		1	
TPM_DisableForceClear	0	1	
TSC_PhysicalPresence	*0*		
TSC_ResetEstablishmentBit			
TPM_GetCapability	1	1	1
TPM_SetCapability			
TPM_GetAuditDigest			
TPM_GetAuditDigestSigned			
TPM_SetOrdinalAuditStatus			
TPM_FieldUpgrade			
TPM_SetRedirection			
TPM_ResetLockValue		1	
TPM_GetCapabilityOwner	*0*	*0*	*0*
TPM_Seal	1	1	1
TPM_Unseal	1	1	1
TPM_UnBind	1	1	1
TPM_CreateWrapKey	1	1	1
TPM_LoadKey2	1	1	1
TPM_GetPubKey	1	1	1
TPM_Sealx	1	1	1
TPM_CreateMigrationBlob		1	
TPM_ConvertMigrationBlob		1	
TPM_AuthorizeMigrationKey		1	
TPM_MigrateKey		*0*	
TPM_CMK_SetRestrictions			

TPM_CMK_ApproveMA			
TPM_CMK_CreateKey			
TPM_CMK_CreateTicket			
TPM_CMK_CreateBlob			
TPM_CMK_ConvertMigration			
TPM_CreateMaintenanceArchive			
TPM_LoadMaintenanceArchive			
TPM_KillMaintenanceFeature			
TPM_LoadManuMaintPub			
TPM_ReadManuMaintPub			
TPM_SHA1Start			
TPM_SHA1Update			
TPM_SHA1Complete			
TPM_SHA1CompleteExtend			
TPM_Sign	1	1	1
TPM_GetRandom	1	1	1
TPM_StirRandom	1	1	1
TPM_CertifyKey	1	1	1
TPM_CertifyKey2			
TPM_CreateEndorsementKeyPair		1	
TPM_CreateRevocableEK			
TPM_RevokeTrust			
TPM_ReadPubek		1	
TPM_OwnerReadInternalPub		1	
TPM_MakeIdentity		1	
TPM_ActivateIdentity		1	
TPM_Extend	1	1	1
TPM_PCRRead	1	1	1
TPM_Quote	1	1	1
TPM_PCR_Reset			
TPM_Quote2	1	1	1
TPM_ChangeAuth	1	1	1
TPM_ChangeAuthOwner		1	
TPM_OIAP	1	1	1
TPM_OSAP	1	1	1

TPM_DSAP			
TPM_SetOwnerPointer			
TPM_Delegate_Manage			
TPM_Delegate_CreateKeyDelegation			
TPM_Delegate_CreateOwnerDelegation			
TPM_Delegate_LoadOwnerDelegation			
TPM_Delegate_ReadTable			
TPM_Delegate_UpdateVerification			
TPM_Delegate_VerifyDelegation			
TPM_NV_DefineSpace			
TPM_NV_WriteValue			
TPM_NV_WriteValueAuth			
TPM_NV_ReadValue			
TPM_NV_ReadValueAuth			
TPM_KeyControlOwner			
TPM_SaveContext			
TPM_LoadContext			
TPM_FlushSpecific	1	1	1
TPM_GetTicks			
TPM_TickStampBlob			
TPM_EstablishTransport	1	1	1
TPM_ExecuteTransport	1	1	1
TPM_ReleaseTransportSigned			
TPM_CreateCounter			
TPM_IncrementCounter	1	1	1
TPM_ReadCounter	1	1	1
TPM_ReleaseCounter			
TPM_ReleaseCounterOwner			
TPM_DAA_Join			
TPM_DAA_Sign			

1
2

5.1.2 Trust Requirements for Use of the Ordinals

Start of informative comment

The following table lists all TPM and MTM ordinals and indicates the level of privilege required to call the ordinal - through the TBB interface or otherwise.

Some ordinals are restricted to use by the engine stakeholder, namely the owner of the corresponding MTM. Thus they can only be called by services that are fully trusted by the stakeholder. The services may be either trusted as delegates, or simply trusted as agents that will receive the owner's authorization data (e.g. by keyboard entry), use it temporarily within an authorization session, and then discard it. In some cases, the ordinals are restricted to the TBB itself, in which case they must not be exposed outside the TBB, as discussed in 5.1.1. The numbers of ordinals in each category are totaled at the top of the table.

End of informative comment.

A number of MTM ordinals MAY be called by any process running in the engine. Other MTM ordinals MUST be restricted only to services that are trusted by the engine's stakeholder, and such services SHOULD always be protected by a RIM. Further ordinals SHALL be restricted to the TBB.

The table below shows which ordinals are in which category, via a "1" in the appropriate cell. Again, ordinals that are shown in normal text are REQUIRED to be supported by either an MRTM or an MLTM or both. Ordinals that are shown in italic text are **not** REQUIRED to be supported by either an MRTM or an MTLM.

Total Required	**25**	**28 / 29**	**3 / 4**
Ordinal Name	**Unrestricted**	**Restricted to Engine Stakeholder (or Proxy, inc TBB)**	**Restricted to TBB alone**
MTM_InstallRIM		1	
MTM_LoadVerificationKey		1	
MTM_LoadVerificationRootKeyDisable			1
MTM_VerifyRIMCert		1	
MTM_VerifyRIMCertAndExtend		1	
MTM_IncrementBootstrapCounter		1	
MTM_SetVerifiedPCRSelection		1 (For MLTM)	1 (For MRTM)
TPM_Startup			1
TPM_SaveState			
TPM_SelfTestFull	1		
TPM_ContinueSelfTest			1
TPM_GetTestResult	1		
TPM_SetOwnerInstall		1	
TPM_OwnerSet		1	
TPM_PhysicalEnable		1	
TPM_PhysicalDisable		1	
TPM_PhysicalSetDeactivated		1	
TPM_SetTempDeactivated		1	

TPM_SetOperatorAuth			
TPM_TakeOwnership		1	
TPM_OwnerClear		1	
TPM_ForceClear		1	
TPM_DisableOwnerClear		1	
TPM_DisableForceClear		1	
TSC_PhysicalPresence			
TSC_ResetEstablishmentBit			
TPM_GetCapability	1		
TPM_SetCapability			
TPM_GetAuditDigest			
TPM_GetAuditDigestSigned			
TPM_SetOrdinalAuditStatus			
TPM_FieldUpgrade			
TPM_SetRedirection			
TPM_ResetLockValue		1	
TPM_GetCapabilityOwner			
TPM_Seal	1		
TPM_Unseal	1		
TPM_UnBind	1		
TPM_CreateWrapKey	1		
TPM_LoadKey2	1		
TPM_GetPubKey	1		
TPM_Sealx	1		
TPM_CreateMigrationBlob		1	
TPM_ConvertMigrationBlob		1	
TPM_AuthorizeMigrationKey		1	
TPM_MigrateKey			
TPM_CMK_SetRestrictions			
TPM_CMK_ApproveMA			
TPM_CMK_CreateKey			
TPM_CMK_CreateTicket			
TPM_CMK_CreateBlob			
TPM_CMK_ConvertMigration			
TPM_CreateMaintenanceArchive			

TPM_LoadMaintenanceArchive			
TPM_KillMaintenanceFeature			
TPM_LoadManuMaintPub			
TPM_ReadManuMaintPub			
TPM_SHA1Start			
TPM_SHA1Update			
TPM_SHA1Complete			
TPM_SHA1CompleteExtend			
TPM_Sign	1		
TPM_GetRandom	1		
TPM_StirRandom		1	
TPM_CertifyKey	1		
TPM_CertifyKey2			
TPM_CreateEndorsementKeyPair		1	
TPM_CreateRevocableEK			
TPM_RevokeTrust			
TPM_ReadPubek		1	
TPM_OwnerReadInternalPub		1	
TPM_MakeIdentity		1	
TPM_ActivateIdentity		1	
TPM_Extend	1		
TPM_PCRRead	1		
TPM_Quote	1		
TPM_PCR_Reset			
TPM_Quote2	1		
TPM_ChangeAuth	1		
TPM_ChangeAuthOwner		1	
TPM_OIAP	1		
TPM_OSAP	1		
TPM_DSAP			
TPM_SetOwnerPointer			
TPM_Delegate_Manage			
TPM_Delegate_CreateKeyDelegation			
TPM_Delegate_CreateOwnerDelegation			
TPM_Delegate_LoadOwnerDelegation			

TPM_Delegate_ReadTable			
TPM_Delegate_UpdateVerification			
TPM_Delegate_VerifyDelegation			
TPM_NV_DefineSpace			
TPM_NV_WriteValue			
TPM_NV_WriteValueAuth			
TPM_NV_ReadValue			
TPM_NV_ReadValueAuth			
TPM_KeyControlOwner			
TPM_SaveContext			
TPM_LoadContext			
TPM_FlushSpecific	1		
TPM_GetTicks			
TPM_TickStampBlob			
TPM_EstablishTransport	1		
TPM_ExecuteTransport	1		
TPM_ReleaseTransportSigned			
TPM_CreateCounter			
TPM_IncrementCounter	1		
TPM_ReadCounter	1		
TPM_ReleaseCounter			
TPM_ReleaseCounterOwner			
TPM_DAA_Join			
TPM_DAA_Sign			

The most efficient way to meet the trust requirements is for the engine's TCB to implicitly identify to the TBB whether the calling entity is a service that is trusted by the engine stakeholder (e.g. because it is protected by a RIM signed by an appropriate RIM_Auth) and hence identify whether the calling entity is allowed access to ordinals marked with a "1" in column 3 of the above table. The TBB SHOULD be configured to accept implicit identification of trusted services via the TCB. In that case:

1. The TBB MUST disallow use of these restricted ordinals if the originator is not identified by the TCB as being a trusted service.
2. There is a design assumption on the TCB that it will protect the interface between the privileged calling entity and the TBB from spoofing and data manipulation.
3. Where the TCB identifies to the TBB that the originator is a trusted service, then the originator does not REQUIRE specific authorization data to prove directly to the TBB that it is a trusted service. However, it MAY need some authData that has to be provided to the underlying MTM to make an ordinal work appropriately, as defined by the MTM or TPM specifications.

Alternatively, the calling entity MAY use explicit authorization data, which we label *tbbAuth*, or any generalized authorization method to establish to the TBB that it is trusted by the engine stakeholder. There is

a design assumption on the TCB that it ensures only trusted services have access to this authorization data. The TBB SHOULD in any case check the integrity of the TCB before allowing the use of restricted ordinals.

5.2 Additional Ordinals Exposed by the TBB

Start of informative comment:

The following table lists additional ordinals which may be exposed by the TBB and indicates the level of privilege required to call the ordinal.

Most ordinals are restricted to use by the engine stakeholder, namely the owner of the corresponding MTM. Thus they can only be called by services that are fully trusted by the stakeholder. The services may be either trusted as delegates, or simply trusted as agents that will receive the owner's authorization data (e.g. by keyboard entry), use it temporarily within an authorization session, and then discard it.

End of informative comment.

The following TBB ordinals are defined in addition to the MTM ordinals. All these ordinals are in general OPTIONAL, however:

1. Ordinals described in Section 5.3 SHOULD be supported where the engine supports secure boot.

2. Ordinals described in Section 5.4 SHOULD be supported where the database of internal RIM certificates is designed so that only the TBB has read access to this database.

3. Ordinals described in Section 5.5 SHOULD be supported where the TBB supports a Watchdog Timer.

4. Ordinals described in Section 5.6 SHOULD be supported where the MTM supports only the minimum requirements on monotonic counters defined in [5].

5. Ordinals described in Section 5.7 SHOULD be supported where the MTM supports owner authorized commands that are delegated to the v*erificationAuth* data.

A number of these TBB ordinals MAY be called by any process running in the engine. Other TBB ordinals MUST be restricted only to services that are trusted by the engine's stakeholder, and such services SHOULD always be protected by a RIM. The table below shows which ordinals are in which category, via a "1" in the appropriate cell.

Total Numbers	**25**	**10**	**15**
Ordinal Name	**Ordinal Number**	**Unrestricted**	**Restricted to Engine Stakeholder or Proxy**
TBB_AddExternalRIMData			1
TBB_RemoveExternalRIMData			1
TBB_InternalizeRIMCertificate			1
TBB_GenerateInternalRIMCertificate			1
TBB_RefreshInternalRIMCertificate			1
TBB_RemoveInternalRIMCertificate			1
TBB_FindInternalRIMCertificate			1
TBB_EnumInternalRIMCertificates			1
TBB_ConfigureWDTMeasurements			1
TBB_SeedASC			1
TBB_IncrementASC			1
TBB_CreateCounter			1
TBB_IncrementCounter		1	

TBB_ReadCounter		1	
TBB_ReleaseCounter		1	
TBB_ReleaseCounterOwner			1
TBB_NV_DefineSpace			1
TBB_NV_WriteValue		1	
TBB_NV_WriteValueAuth		1	
TBB_NV_ReadValue		1	
TBB_NV_ReadValueAuth		1	
TBB_ProtectVirtualCountersAndStorage			1
TBB_OIAP		1	
TBB_OSAP		1	
TBB_DSAP		1	

The most efficient way to meet the trust requirements is for the engine's TCB to implicitly identify the calling entity to the TBB as a service which is trusted by the engine stakeholder. Alternatively, the calling entity MAY use its own explicit authorization data (i.e. *tbbAuth*) or any generalized authorization method to establish to the TBB that it is a trusted service.

In the case of explicit authorization data, the calling entity SHOULD use TBB_OIAP, TBB_OSAP or TBB_DSAP with *tbbAuth* to establish its privileges to the TBB: see 5.7 below. This approach makes it unnecessary to store or seal a separate copy of the *ownerAuth* secret for use by the TBB outside the MTM. Instead only the distinct *tbbAuth* is stored within the TBB (or sealed to the TBB) for use outside the MTM.

To support a number of the ordinals, the TBB is REQUIRED to hold a database containing (for example) external RIM data or internal RIM certificates. The TBB SHOULD always have access to sufficient storage space to hold this data, and it is assumed in the ordinal design that (unlike in a TPM or MTM) there is not a tight limit on storage space. The TBB SHOULD be able to store its data in an integrity-protected (and optionally encrypted) form using normal non-volatile memory. If nevertheless the operation of an ordinal would cause the TBB to exceed a size quota, then it MAY respond with a TPM_SIZE error.

5.3 Communication with Secure RIM Conversion Agent

Start of informative comment:

The use cases for ordinals "TBB_AddExternalRIMData", and "TBB_RemoveExternalRIMData" are adding and removing external RIM_Certs, RIM_Auth_Certs and Validity_Lists to an external RIM database, and checking their validity before adding. When adding, note that the handles of existing database objects which this RIM data appears to supercede are identified and returned. The calling entity should in general attempt to remove redundant objects. On removing, note that it is not possible to remove the latest known validity list signed by any RIM_Auth: this is a security measure to prevent rollback attacks.

The use case for "TBB_InternalizeRIMCertificate" is asking a RIM Conversion Agent to create Internal RIM_Certs from valid external RIM_Certs. Alternatively, the RIM Conversion Agent is asked to make a current state measurement and directly create an Internal RIM_Cert from that measurement via "TBB_GenerateInternalRIMCertificate". (A use-case for this can be found in [9] Section 7.2.5 "Consistency between Installation and Launch"). Note that the creation of an Internal RIM_Cert is a highly sensitive operation, so the TBB may need to perform detailed security checks before allowing use of these ordinals.

The use cases for ordinals "TBB_RefreshInternalRIMCertificate" and "TBB_RemoveInternalRIMCertificate" arise when creating a new generation of internal RIM Certificates to go with a new counterRIMProtect value. As discussed in [5] Section 7.2, internal RIM Certificates which need to remain valid should be refreshed ***first*** and once they have been refreshed, obsolete internal RIM Certificates should then be removed. This avoids problems that could arise if the refresh/remove operations are interrupted. The refresh re-signs a valid existing internal RIM Certificate, using the current counterRIMProtect + 1, as described in MTM_InstallRIM.

There are two possible models by which calling entities access the external RIM database. They may have direct read access to the TBB's own database, or they may have read and write access to a copy of the database which is maintained outside the TBB, for example within the Trusted Services API. Where a database copy is used, then the copy must be kept synchronized with the TBB's database, including recording the handles assigned by the TBB database on each use of "TBB_AddExternalRIMData".

The following type and tag definitions are re-used from [9].

End of informative comment.

typedef BYTE UTCTIME[13];

TPM_TAG_VERIFICATION_KEY	0x0301	TPM_VERIFICATION_KEY
TPM_TAG_RIM_CERTIFICATE	0x0302	TPM_RIM_CERTIFICATE
TPM_TAG_RIM_AUTH_VALIDITY_LIST	0x0305	TPM_RIM_AUTH_VALIDITY_LIST
TPM_TAG_RIM_VALIDITY_LIST	0x0306	TPM_RIM_VALIDITY_LIST

At engine creation, or at latest when ownership of an engine is first established, then the TBB SHALL contain databases of external and internal RIM data, with the external database containing at minimum the RVAI root public key as a TPM_VERIFICATION_KEY. This RVAI entry in the database SHALL have a recorded validityLimit of type UTCTIME. The validityLimit SHALL be set so that either all its bytes are equal to 0xFF or it contains the latest date and time at which the engine is permitted by the owner to remain operational.

On each increment of an MTM monotonic counter, the TBB SHALL re-validate all TPM_VERIFICATION_KEY and TPM_RIM_CERTIFICATE entries in the databases whose counterReference->counterSelection refers to the counter just incremented. Where an object's counterReference->counterValue is now less than the new value of the monotonic counter, the TBB SHALL set all bytes of the object's validityLimit to 0x00. Where a TPM_VERIFICATION_KEY is invalidated in this way, the TBB SHALL similarly invalidate any other database objects signed by that key, or signed by any descendant of that key, by setting their validityLimits to all 0x00.

1 ### 5.3.1 TBB_AddExternalRIMData

2 ## Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	Tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TBB_ORD_AddExternalRIMData
4	2	2S	2	TPM_TAG	rimDataTag	Type of RIM data being added
5	4	3S	4	UINT32	rimDataSize	Size of following RIM data
6	<>	4S	<>	BYTE []	rimData	The actual RIM data
7	4	5S	4	UINT32	extensionAddressSize	Size of extension start and end addresses
8	<>	6S	<>	BYTE []	extensionStartAddress	Start of extension data that is to be stored with the RIM data.
9	<>	7S	4	UINT32	extensionEndAddress	End of extension data that is to be stored with the RIM data.
10	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TBB to cover inputs
11	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
12	1	4H1	1	BOOL	continueAuthSession	The continue use flage for the authorization session handle.
13	20			TPM_AUTHDATA	authData	OPTIONALLY the authorization session digest for inputs. HMAC key: tbbAuth

3 ## Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	Tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TBB_ORD_ AddExternalRIMData
4	4	3S	4	UINT32	rimDataHandle	Handle for the stored RIM data
5	4	4S	4	UINT32	matchingDataSize	Size of following RIM handles
6	<>	5S	<>	BYTE[]	matchingDataHandles	Handles for matching RIM data already in database
7	1	6S	1	BOOL	validityStatus	Is the RIM Data valid?
8	13	7S	13	UTCTIME	validityLimit	Last date and time at which the RIM Data is valid
9	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TBB to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
10	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
11	20			TPM_AUTHDATA	resAuth	OPTIONALLY The authorization session digest for the returned parameters. HMAC key: tbbAuth

4

Action

The TBB SHALL perform the following steps:

1. If the TCB has **not** identified to the TBB that the originator is a trusted service, validate the command authorization parameters based on *tbbAuth*. If the validation fails then return TPM_AUTHFAIL.
2. Look at the rimDataTag and use it to type-check the rimData:
    a. If the tag has value TPM_TAG_VERIFICATION_KEY, rimData should be a TPM_VERIFICATION_KEY
    b. If the tag has value TPM_TAG_RIM_CERTIFICATE, rimData should be a TPM_RIM_CERTIFICATE
    c. If tag is TPM_TAG_RIM_AUTH_VALIDITY_LIST, rimData should be a TPM_RIM_AUTH_VALIDITY_LIST
    d. If tag is TPM_TAG_RIM_VALIDITY_LIST, rimData should be a TPM_RIM_VALIDITY_LIST

    If the rimData does not have the appropriate syntax then return TPM_BAD_PARAMETER.
3. If extensionAddressSize != 0, then first check that the rimDataTag is TPM_TAG_VERIFICATION_KEY or TPM_TAG_RIM_CERTIFICATE, and set extensionData equal to the data between extensionStartAddress and extensionEndAddress inclusive. Checknext that the hash of the extensionData using an acceptable hash function as defined in [5] matches the extensionDigestSize and extensionDigestData within the rimData structure. If not then return TPM_BAD_PARAMETER.
4. Check if the rimData matches external RIM data that is already stored within the TBB database:
    a. For a TPM_RIM_CERTIFICATE, check if the label matches the label of any existing external TPM_RIM_CERTIFICATE(s).
    b. For a TPM_VERIFICATION_KEY check if the myId matches the myId of any existing TPM_VERIFICATION_KEY(s).
    c. For a TPM_RIM_AUTH_VALIDITY_LIST check if the signer_id matches the signer_id of any existing TPM_RIM_AUTH_VALIDITY_LIST(s).
    d. For a TPM_RIM_VALIDITY_LIST check if the signer_id matches the signer_id of any existing TPM_RIM_AUTH_VALIDITY_LIST(s).

    Create a list (possibly empty) of the handles of matching RIM data, and set matchingDataHandles equal to a concatenation of handles on that list. Set matchingDataSize = 4 times the number of entries on the list.
5. If this exact rimData is already stored within the TBB's database of external RIM data, then
    a. Set rimDataHandle = the handle already assigned to it within that database
    b. Set validityLimit = the validity limit already recorded within the database
    c. Set validityStatus = TRUE if the object is currently valid (i.e. its recorded validityLimit in the database is at least the current date and time) and = FALSE otherwise.
    d. If the command was authorized, then construct the output authorization parameters.
    e. Return TPM_SUCCESS.
6. Otherwise, generate a fresh value that is not used for anything already in the database and set rimDataHandle equal to that value.
7. Attempt to validate the rimData using the existing external RIM data stored in the database.
    a. If the rimData is a TPM_VERIFICATION_KEY or TPM_RIM_CERTIFICATE, check that its parentId corresponds to a TPM_VERIFICATION_KEY that is marked as currently valid within the database

b. If the rimData is a TPM_RIM_AUTH_VALIDITY_LIST or a TPM_RIM_VALIDITY_LIST, check that its signer_id corresponds to a TPM_VERIFICATION_KEY marked as currently valid within the database i.e. its recorded validityLimit in the database is at least the current date and time.

c. Once the signing key is identified as valid, check that the signature on the rimData is correctly formed using that key, and that the key is authorized to sign that rimData. This last check requires examination of the usageFlags of the stored signing key and any usage restriction lists in the stored extension data bound to the stored signing key.

d. If the rimData is a TPM_RIM_VALIDITY_LIST or a TPM_RIM_AUTH_VALIDITY_LIST, then check that its validFrom date is at least as recent as that of the last stored (respectively) TPM_RIM_VALIDITY_LIST or TPM_RIM_AUTH_VALIDITY_LIST which is signed by the same signer_id. (If there is no corresponding stored list, then bypass this check). Also check that the validTo date of the rimData has not yet passed.

e. If the rimData is a TPM_VERIFICATION_KEY, and its parentId is marked as signing RIM_Auth Validity lists, then check that there is at least one TPM_RIM_AUTH_VALIDITY_LIST in the database marked as currently valid, whose signer_id matches that parentId, and that the myId field in the rimData appears on the most recent such valid TPM_RIM_AUTH_VALIDITY_LIST (i.e. the one with the latest validFrom date).

f. If the rimData is a TPM_RIM_CERTIFICATE, and the parentId of the rimData is marked as signing RIM Validity lists, then check that there is at least one TPM_RIM_VALIDITY_LIST in the database marked as currently valid, and whose signer_id matches that parentId and that the serial number constructed from the label and version fields of the of rimData appears on the most recent such valid TPM_RIM_VALIDITY_LIST (i.e. the one with the latest validFrom date).

g. If the rimData is a TPM_VERIFICATION_KEY or TPM_RIM_CERTIFICATE then check that the rimData's referenceCounter->counterValue is not less than the current value of the MTM counter indicated by referenceCounter->counterSelection.

If all this validation succeeds, then set validityStatus = TRUE. Otherwise set validityStatus = FALSE.

8. If the validityStatus == TRUE then set validityLimit = the earliest date at which this rimData's validity expires. Calculate this date as follows:

   a. If the rimData is a TPM_RIM_CERTIFICATE or a TPM_RIM_VALIDITY_LIST, then identify the signing TPM_VERIFICATION_KEY of the rimData as key1, and – if key1 is marked as signing TPM_RIM_VALIDITY_LISTs - find the most recent valid TPM_RIM_VALIDITY_LIST signed by key1.

   b. If the rimData is a TPM_VERIFICATION_KEY or a TPM_RIM_AUTH_VALIDITY_LIST, then identify the signing TPM_VERIFICATION_KEY key1 of the rimData and – if key1 is marked as signing TPM_RIM_AUTH_VALIDITY_LISTs - find the most recent valid TPM_RIM_AUTH_VALIDITY_LIST signed by key1.

   c. Set validityLimit to the minimum of the validTo date on the selected validity List, and the existing validityLimit of key1. If there is no selected validity list then set validityLimit to the existing validityLimit of key1.

   If the validityStatus == FALSE, then set all bytes of validityLimit to 0x00.

9. Add the rimData to the database, indexed by its handle. Also store the calculated validityLimit alongside the rimDatait, and if requested, store the extensionData is defined, store that as well (or at least store the extensionStartAddress and extensionEndAddress).

10. If the rimData has validityStatus == TRUE and is a Validity List, then use it to (recursively) update the validity limits of all other database entries whose own parentId matches the rimData's signer_id (or recursively, whose parentId or signer_id is a descendant of that signer_id). Update a validity limit to all 0x00 whenever a database object becomes invalid because that object (or its ancestor) would be expected to appear on the Validity List but no longer appears there.

11. If the rimData has validityStatus == TRUE and is a TPM_VERIFICATION_KEY, then use it to (recursively) revalidate – and set the validityLimit for – all other database entries whose own parentId or signer_id matches the rimData's myId (or recursively, whose parentId or signer_id is a descendant of that myId).
12. If the command was authorized, then construct the output authorization parameters.
13. Return TPM_SUCCESS.

5.3.2 TBB_RemoveExternalRIMData

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	Tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TBB_ORD_RemoveExternalRIMData
4	4	2S	4	UINT32	handle	Handle of external RIM data to remove
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TBB to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flage for the authorization session handle.
8	20			TPM_AUTHDATA	authData	OPTIONALLY the authorization session digest for inputs. HMAC key: tbbAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TBB_ORD_ RemoveExternalRIMData
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TBB to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	OPTIONALLY The authorization session digest for the returned parameters. HMAC key: tbbAuth

Action

The TBB SHALL perform the following steps:

1. If the TCB has **not** identified to the TBB that the originator is a trusted service, validate the command authorization parameters based on *tbbAuth*. If the validation fails then return TPM_AUTHFAIL.
2. Look up the external RIM data by handle in the TBB database.
    a. If no RIM data exists with that handle, then return TPM_BAD_PARAMETER.
    b. If the RIM data is of type TPM_RIM_AUTH_VALIDITY_LIST check if it has a validityLimit > 0. If it does, check that there is another TPM_RIM_AUTH_VALIDITY_LIST in the database which
        i. Is signed by the same key referenced by the signer_id field of the RIM data
        ii. Also has a validityLimit > 0, and
        iii. Has a later validFrom field than the RIM data.

If there is no such list in the database, then return TPM_FAIL.

c. If the RIM data is of type TPM_RIM_VALIDITY_LIST check to see if it has a validityLimit > 0. If so, check that there is another TPM_RIM_VALIDITY_LIST in the database which

i. Is signed by the same key referenced by signer_id field of the RIM data

ii. Also has a validityLimit > 0, and

iii. Has a later validFrom field than the RIM data.

If there is no such list in the database, then return TPM_FAIL.

d. Otherwise delete the external RIM data from the database, together with any stored extension data where applicable.

3. If the command was authorized, then construct the output authorization parameters.
4. Return TPM_SUCCESS.

5.3.3 TBB_InternalizeRIMCertificate

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	Tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TBB_ORD_InternalizeRIMCertificate
4	4	2S	4	UINT32	handle	Handle of external RIM certificate to internalize
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TBB to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flage for the authorization session handle.
8	20			TPM_AUTHDATA	authData	OPTIONALLY the authorization session digest for inputs. HMAC key: tbbAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TBB_ORD_InternalizeRIMCertificate
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TBB to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	OPTIONALLY The authorization session digest for the returned parameters. HMAC key: tbbAuth

Action

The TBB SHALL perform the following steps:

1. If the TCB has **not** identified to the TBB that the originator is a trusted service, validate the command authorization parameters based on *tbbAuth*. If the validation fails then return TPM_AUTHFAIL.
2. Look up the handle in the TBB database of **external** RIM data :
    a. Check that an entry, rimData, exists with that handle
    b. Check that rimData is of type TPM_RIM_CERTIFICATE
    c. Check that rimData is currently marked as valid in the database
    d. Read the label and rimVersion fields of rimData and check there is no existing entry in the TBB database of **internal** RIM certificates with the same label and rimVersion

    If any of these checks fail, then return TPM_BAD_PARAMETER.
3. **(Optional)** Re-validate rimData by re-building the chain up to the RVAI, re-verifying the integrity and authorization of all entries in the chain, and re-validating all entries in the chain with the most recent Validity List (where appropriate). If this re-validation fails, then return TPM_AUTH_FAIL.
4. **(Optional)** Perform other system integrity checks, where relevant, such as checking the current state of PCRs, executing PRMVA measurements and verifications, and/or performing MTM_LoadVerificationKey with the RVAI public key to ensure it has not been corrupted or replaced. If any of these system integrity checks fail then return TPM_FAIL and trigger a transition to a "FAILED" state.
5. Attempt to execute MTM_InstallRIM using rimData as the rimCertIn parameter. If this command fails, then return TPM_FAIL.
6. Write the new rimCertOut to the TBB database of internal RIM Certificates, indexed by label and version. Ensure that all extension data associated with rimData is also associated with rimCertOut.
7. If the command was authorized, then construct the output authorization parameters.
8. Return TPM_SUCCESS.

1 5.3.4 **TBB_GenerateInternalRIMCertificate**

2 **Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	Tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TBB_ORD_GenerateInternallRIMCertificate
4	4	2S	4	UINT32	rimTemplateSize	Size of following RIM template data
5	<>	3S	<>	TPM_RIM_CERTIFICATE	rimTemplateData	Template for the internal RIM Cert to be Created
6	4	4S	4	UINT32	extensionAddressSize	Size of extension start and end addresses
7	<>	5S	<>	BYTE []	extensionStartAddress	Start of extension data that is to be stored with the RIM data.
8	<>	6S	<>	BYTE []	extensionEndAddress	End of extension data that is to be stored with the RIM data.
9	4	7S	4	UINT32	segmentAddressSize	Size of segment start and end addresses
10	<>	8S	<>	BYTE []	segmentStartAddress	Start address of segment for reference measurement
11	<>	9S	<>	BYTE []	segmentEndAddress	End address of segment for reference measurement
12	4	10S	4	UINT32	addAuthAddressSize	Size of additional authorization start and end addresses
13	<>	11S	<>	BYTE []	addAuthStartAddress	Start of extra authorization data for creating internal RIM Cert
14	<>	12S	<>	BYTE []	addAuthEndAddress	End of extra authorization data for creating internal RIM Cert
15	4	13S	4	UINT32	Handle	Handle of external RIM data to verify authorization
16	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TBB to cover inputs
17	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
18	1	4H1	1	BOOL	continueAuthSession	The continue use flage for the authorization session handle.
19	20			TPM_AUTHDATA	authData	OPTIONALLY the authorization session digest for inputs. HMAC key: tbbAuth

3 **Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TBB_ORD_GenerateInternallRIMCertificate
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TBB to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	OPTIONALLY The authorization session digest for the returned parameters. HMAC key: tbbAuth

4

Action

The TBB SHALL perform the following steps:

1. If the TCB has **not** identified to the TBB that the originator is a trusted service, validate the command authorization parameters based on *tbbAuth*. If the validation fails then return TPM_AUTHFAIL.
2. Parse the rimTemplateData:
    a. Check that it has the appropriate syntax for a TPM_RIM_CERTIFICATE
    b. Read the label and rimVersion fields and check there is no existing entry in the TBB database of **internal** RIM certificates with the same label and rimVersion
    c. If extensionAddressSize != 0, then first set extensionData equal to the data between extensionStartAddress and extensionEndAddress inclusive. Checkcheck that the hash of the extensionData using an acceptable hash function as defined in [5] matches the extensionDigestSize and extensionDigestData within the rimTemplateData.

    If any of these checks fail, then return TPM_BAD_PARAMETER.
3. If handle != NULL, then look-up the handle in the TBB database of **external** RIM data :
    a. Check that an entry, rimData, exists with that handle
    b. Check that rimData is currently marked as valid in the database

    If any of these checks fail, then return TPM_BAD_PARAMETER.
4. **(Optional)** Re-validate rimData by re-building the chain up to the RVAI, re-verifying the integrity and authorization of all entries in the chain, and re-validating all entries in the chain with the most recent Validity List (where appropriate). If this re-validation fails, then return TPM_AUTH_FAIL.
5. If addAuthAddressSize != 0, then first set addAuthData equal to the data between addAuthStartAddress and addAuthEndAddress inclusive.**(Optional)** Check that addAuthData and/or rimData correctly authorize the generation of an internal RIM Certificate composed of rimTemplateData, the reference measurement indicated by startAddress and endAddress, and the extensionData. If this authorization fails, then return TPM_AUTH_FAIL.

Start of informative comment:

The exact mechanism above is manufacturer specific, but there are a number of mechanisms by which additional authorization data may be provided. For example, the addAuthData could be signed by a RIM Auth, and the rimData is then just the TPM_VERIFICATION_KEY needed to verify the signature.

Or the rimData could be a special external RIM Certificate, which indicates the expected PCR state just prior to a generating the internal RIM Cert, together with an expected state of the "Normal" component of the RIM Conversion agent at this stage. The TBB then checks everything is in order by measuring this Normal component and attempting MTM_VerifyRIMCertAndExtend using rimData.

A third option: the rimData is bound to additional information via its extensionDigestData. This additional information is then used in conjunction with the addAuthData to authorize the creation of the internal RIM Certificate. Such additional bound information could for instance include a set of instructions for unpacking the new object when delivered to the engine, with addAuthData then a log of the unpacking operation performed by the Normal component (in accordance with those instructions). This log demonstrates that the new object has been loaded to the address range indicated by startAddress and endAddress ready for post-install measurement.

End of informative comment.

6. Perform the measurement of the address segment indicated by startAddress and endAddress and set rimTemplateData->measurementValue to the result of that measurement.

7. **(Optional)** Perform other system integrity checks, where relevant, such as checking the current state of PCRs, executing PRMVA measurements and verifications. If any of these system integrity checks fail then return TPM_FAIL and trigger a transition to a "FAILED" state.
8. **(Optional)** Further transform rimTemplateData and associated extensionData as necessary, in accordance with TBB design and/or the authorization instructions in addAuthData or rimData.

Start of informative comment:

For example: customize the PCR pre-condition rimTemplateData->state to fit the PCR pre-condition indicated in rimData, or customize it to fit the expected state of this given engine instance.

Or add to the extensionData, as needed, e.g. by merging with the extensions already associated with rimData. Update rimTemplateData->extensionDigestData with a hash of the new extensionData.

End of informative comment.

9. Attempt to execute MTM_InstallRIM using rimTemplateData as the rimCertIn parameter. If this command fails, then return TPM_FAIL.
10. Write the new rimCertOut to the TBB database of **internal** RIM Certificates, indexed by label and version. Ensure that the final extensionData is also associated with that rimCertOut.
11. If the command was authorized, then construct the output authorization parameters.
12. Return TPM_SUCCESS.

5.3.5 TBB_RefreshInternalRIMCertificate

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	Tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TBB_ORD_RefreshInternalRIMCertificate
4	12	2S	12	RTV_RIMCERTSERIAL	serialINumber	Serial number of internal RIM Cert to refresh
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TBB to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flage for the authorization session handle.
8	20			TPM_AUTHDATA	authData	OPTIONALLY the authorization session digest for inputs. HMAC key: tbbAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TBB_ORD_ RefreshInternalRIMCertificate
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TBB to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	OPTIONALLY The authorization session digest for the returned parameters. HMAC key: tbbAuth

Action

The TBB SHALL perform the following steps:

1. If the TCB has **not** identified to the TBB that the originator is a trusted service, validate the command authorization parameters based on *tbbAuth*. If the validation fails then return TPM_AUTHFAIL.
2. Look up the RIM Certificate by serial number in the TBB database of internal RIM Certificates.
    a. Check that an entry, rimCert, exists with that serial number
    b. Check rimCert's referenceCounter->counterSelection == MTM_COUNTER_SELECT_RIMPROTECT

   If any of these checks fail, then return TPM_BAD_PARAMETER.
3. Attempt to verify rimCert using MTM_VerifyRIMCert with rimKey set to NULL. If the verification fails, then return TPM_FAIL.

4. **(Optional)** Perform other system integrity checks, where relevant, such as checking the current state of PCRs, executing PRMVA measurements and verifications. If any of these system integrity checks fail then return TPM_FAIL and trigger a transition to a "FAILED" state.
5. Attempt to execute MTM_InstallRIM using rimCert as the rimCertIn parameter. If this command fails, then return TPM_FAIL.
6. Write the new rimCertOut to the TBB database of internal RIM Certificates, indexed by label and version, overwriting rimCert in the process. Retain all extension data associated with rimCert.
7. If the command was authorized, then construct the output authorization parameters.
8. Return TPM_SUCCESS.

5.3.6 TBB_RemoveInternalRIMCertificate

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	Tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TBB_ORD_RemoveInternalRIMCertificate
4	12	2S	12	RTV_RIMCERTSERIAL	seriallNumber	Serial number of internal RIM Cert to remove
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TBB to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flage for the authorization session handle.
8	20			TPM_AUTHDATA	authData	OPTIONALLY the authorization session digest for inputs. HMAC key: tbbAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TBB_ORD_ RemoveInternalRIMCertificate
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TBB to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	OPTIONALLY The authorization session digest for the returned parameters. HMAC key: tbbAuth

Action

The TBB SHALL perform the following steps:

1. If the TCB has **not** identified to the TBB that the originator is a trusted service, validate the command authorization parameters based on *tbbAuth*. If the validation fails then return TPM_AUTHFAIL.
2. Look up the RIM Certificate by serial number in the TBB database. If no internal RIM certificate exists with that serial number, then return TPM_BAD_PARAMETER.
3. Where the referenceCounter->counterSelection == MTM_COUNTER_SELECT_RIMPROTECT in this RIM Certificate, set cntValRef = referenceCounter->counterValue i.e. the counterRIMProtect value stored in this RIM Certificate. Also set cntValAct to the result of TPM_ReadCounter of counter MTM_PERMANENT_DATA->counterRimProtectId.
4. Delete the RIM Certificate from the database. Also delete any extension Data stored with the RIM Certificate.
5. If cntValRef == cntValAct then perform TPM_IncrementCounter with the countID set to MTM_PERMANENT_DATA->counterRimProtectId. If this fails, then return TPM_BAD_COUNTER.
6. If the command was authorized, then construct the output authorization parameters.
7. Return TPM_SUCCESS.

5.4 Look-up from a Secure RIM Database

Start of informative comment:

The use case for these ordinals is a model where the TBB alone has read access to a database of internal RIM certificates. This might be easier to arrange and more secure than ensuring that only the TBB has write access, while the rest of the platform has read access. "TBB_FindInternalRIMCertificate" retrieves an internal RIM Cert by label and – if that is ambiguous – by version number. "TBB_EnumInternalRIMCertificates" returns a list of all serial numbers for internal RIM Certs.

The type definition of a RIM Cert Serial Number is re-used from [9], Section 5.2.4.

End of informative comment.

typedef BYTE RTV_RIMCERTSERIAL[12];

The RIM_Cert SerialNumber is a sequence of 12 bytes formed as follows. The first 8 bytes in the sequence are the label field in the RIM_Cert (of type BYTE[8]). The final four bytes in the sequence are the rimVersion field in the RIM_Cert (of type UINT32), ordered according to the big-endian convention.

For example: if label == 0xFFEEDDCCBBAA9988 and rim_version == 0x00000002, then the combined twelve byte serial number == 0xFFEEDDCCBBAA998800000002

5.4.1 TBB_FindInternalRIMCertificate

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	Tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TBB_ORD_FindInternalRIMCertificate
4	8	2S	8	BYTE[8]	label	Label of internal RIM Cert to retrieve
5	4	3S	4	UINT32	rimVersion	Version of internal RIM Cert to retrieve in case of ambiguity
6	1	4S	1	BOOL	includeExtensions	Retrieve extension data as well?
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TBB to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	The continue use flage for the authorization session handle.
10	20			TPM_AUTHDATA	authData	OPTIONALLY the authorization session digest for inputs. HMAC key: tbbAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TBB_ORD_ FindInternalRIMCertificate
4	4	3S	4	UINT32	rimCertSize	Size of rimCert data
5	<>	4S	<>	TPM_RIM_CERTIFICATE	rimCertOut	An internal RIM certificate
6	4	5S	4	UINT32	extensionAddressSize	Size of extension start and end addresses
7	<>	6S	<>	BYTE []	extensionStartAddress	Start of extension data that is associated with the rimCert.
8	<>	7S	<>	BYTE []	extensionEndAddress	End of extension data that is associated with the rimCert
9	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TBB to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
10	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
11	20			TPM_AUTHDATA	resAuth	OPTIONALLY The authorization session digest for the returned parameters. HMAC key: tbbAuth

Action

The TBB SHALL perform the following steps:

1. If the TCB has **not** identified to the TBB that the originator is a trusted service, validate the command authorization parameters based on *tbbAuth*. If the validation fails then return TPM_AUTHFAIL.
2. Look up the RIM Certificate by label in the TBB database.
    a. If no certificate exists in the database with that label, then return TPM_BAD_PARAMETER.
    b. If a unique certificate exists in the database with that label, and rimVersion = 0xFFFFFFFF, then set rimCertOut equal to that unique certificate.
    c. If there are multiple certificates in the database with that label, or rimVersion is not 0xFFFFFFFF, check that there is a certificate which matches both the label and the rimVersion. If there is a unique match then set rimCertOut equal to that unique certificate.
    d. Otherwise return TPM_BAD_PARAMETER.
3. If includeExtensions == TRUE, and extensions have been associated and stored with the RIM Certificates, then set extensionAddressSize, extensionStartAddress and extensionEndAddress to point to the associated and stored extension data. Otherwise set extensionAddressSize = 0.
4. If the command was authorized, then construct the output authorization parameters.
5. Return TPM_SUCCESS.

5.4.2 TBB_EnumInternalRIMCertificates

Start of informative comment:

If there are a large number of RIM certificates, then the amount of memory available for the outgoing operands may be insufficient, thus a start index needs to be specified when calling this API. By checking the return code for a TPM_SIZE error and dividing the returned serial list size by 12, the index to restart at can be determined.

If another process updates the RIM Certificate database between calls to this API, the behavior is undefined.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TBB_ORD_EnumInternalRIMCertificates
4	4	2S	4	UINT32	startIndex	0-based index of first certificate to retrieve
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TBB to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flage for the authorization session handle.
8	20			TPM_AUTHDATA	authData	OPTIONALLY the authorization session digest for inputs. HMAC key: tbbAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TBB_ORD_ EnumInternalRIMCertificates
4	4	3S	4	UINT32	serialListSize	Size in bytes of following list of serial numbers
5	<>	4S	<>	RTV_RIMCERTSERIAL []	serialList	List of serial numbers of internal RIM Certs
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TBB to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20			TPM_AUTHDATA	resAuth	OPTIONALLY The authorization session digest for the returned parameters. HMAC key: tbbAuth

Action

The TBB SHALL perform the following steps:

1. If the TCB has **not** identified to the TBB that the originator is a trusted service, validate the command authorization parameters based on *tbbAuth*. If the validation fails then return TPM_AUTHFAIL.
2. If startIndex is greater or equal to the number of internal RIM Certificates in the tables then return TPM_BAD_PARAMETER.
3. Set serialList = concatenated list of serial numbers of internal RIM Certificates in the database starting from the startIndex, and up to either all the records in the database or the maximum allowable size of the serialList. Set serialListSize = 12 times the number of entries on the list.
4. If the command was authorized, then construct the output authorization parameters.
5. If not all the serial numbers were copied to the output buffer then return TPM_SIZE.
6. Return TPM_SUCCESS.

5.5 Communication with PRMVA/Watchdog Timer

Start of informative comment:

The use-case for these ordinals is the mandatory functionality from the Mobile Reference Architecture specification, conditional on implementing TCG_Reactive capabilities as a Watchdog Timer. The functionality relevant for "TBB_ConfigureWDTMeasurements" is as follows:

1. Upon an event failure the PRMVA must generate a Mandatory Error Response.
2. The Mandatory Error Response:
    a. Must cause access-denial to all cell phone telephony resources and services (where these are available to the Engine), with the possible exception of emergency assistance services when the integrity of those services can be assured.
    b. Must disable or block MTM functionality until the next time the engine boots.
3. The PRMVA must be able to detect when it has not been called within a (configurable) time period. If the time period elapses without a call, the PRMVA must treat this as an event failure.

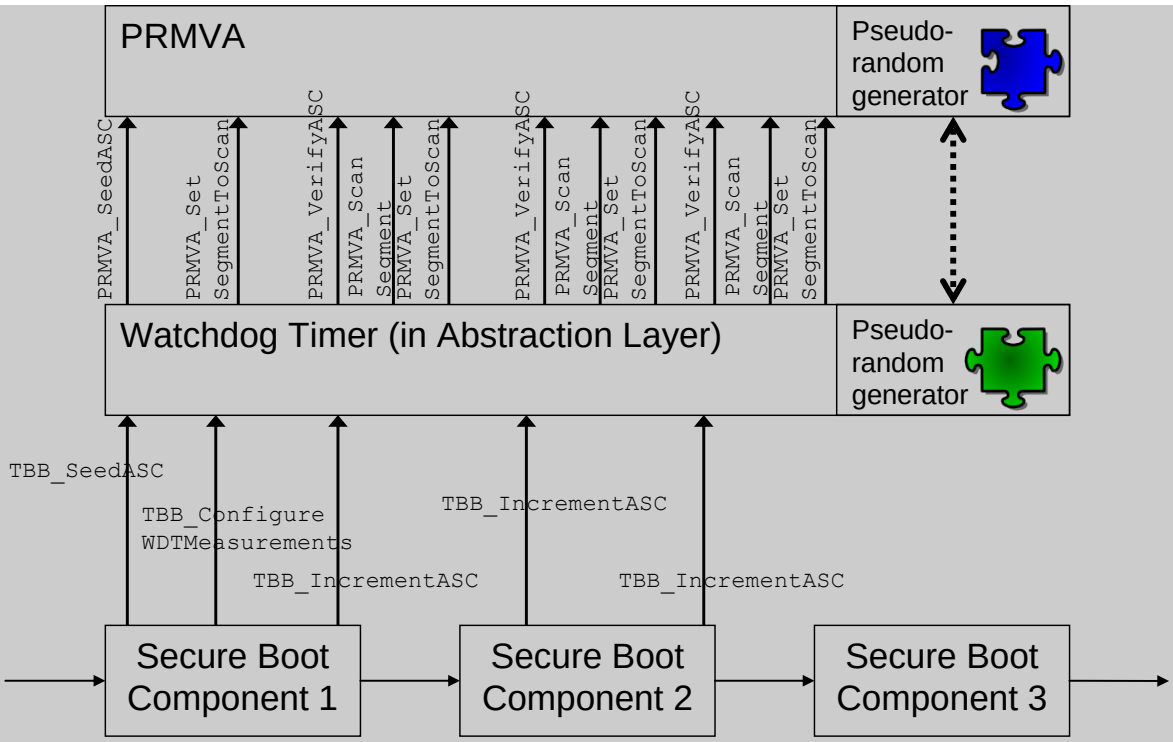
Where the PRMVA is used to measure other engine code, the secure integrity measurement:
1. Must have configuration data which can be locked down on each boot-cycle until the next engine boot independent of run-time host software
2. Must have scan configuration data for each segment which includes control of:
    a. The address range of each memory segment to be measured
    b. The expected/reference value of each segment's measurement
    c. The rate at which memory is to be scanned
    d. The number of memory segments to be scanned

The segment for which configuration data is supplied is identified by a 16 byte "segmentId" field, which may for convenience be structured as a GUID. To aid tracking between boot time and subsequent run-time measurements, some bytes of this field may be used to contain the 8 byte label of a corresponding RIM Cert.

The functionality relevant for "TBB_SeedASC" and "TBB_IncrementASC" is as follows:

The Algorithm Sequence Checker:
1 Must have a pseudo-random sequence which is started on each boot-cycle, and uniquely seeded, with the seed distributed to both the calling function and the PRMVA.
2 May set a configurable pseudo-random number equation (i.e. feedback taps) on each boot-cycle.
3 Must generate a new pseudo-random sequence state/count upon receiving an INCREMENT command
4 Must receive and validate a number [ExpectedState] with every INCREMENT command against its new pseudo-random sequence state/number
5 *Must generate a PRMVA Mandatory Error Response upon a validation FAILURE.*

A design for the interaction between the Abstraction Layer WDT and the PRMVA is as follows:



End of informative comment.

During each boot cycle, the PRMVA SHOULD in general be called multiple times.

The first execution of the PRMVA SHALL be triggered by *TBB_SeedASC*, which determines the pseudo-random sequence that will be used on subsequent calls of the PRMVA this boot cycle. *TBB_SeedASC* SHOULD only be called **once** on each engine boot cycle, and SHOULD be called by the TCB during or shortly after the completion of the secure boot process. Attempts to call it again MUST be rejected by the TBB.

The REQUIRED input parameters to *TBB_SeedASC* are a seed fragment (a length and random value) and a manufacturer-specific identifier for the proposed pseudo-random algorithm. The REQUIRED output parameters are the remainder of the seed (a length and random value) and the agreed algorithm identifier accepted by the TBB. This mechanism ensures that both the TCB and the TBB MAY contribute random data to the seed, and that the TBB selects its preferred algorithm. The input and output seed values MAY also be encrypted (e.g. to a TPM key or a pre-shared secret). However, if there is already a secure channel in place between the TCB and the TBB then further encryption is NOT REQUIRED.

Subsequent executions of the PRMVA SHALL then be triggered by *TBB_IncrementASC*. If TBB_IncrementASC is called with an incorrect sequence number, then this MUST be treated by the PRMVA as an integrity failure.

In any given boot cycle, the WDT configuration data MUST be locked down as soon as *TBB_SeedASC* has been successfully called. The configuration data MAY be changed after boot and before this lock down, or MAY be changed subsequently – either way by using *TBB_ConfigureWDTMeasurements* - but changes MUST only take effect after the next successful call of *TBB_SeedASC*.

After each run of the PRMVA, the WDT MUST use the configuration data for this boot cycle to identify:

- Which segment or segments the PRMVA will scan on its *next* execution during this boot cycle
- A maximum allowed timeout period that is able to elapse before the next execution.
- A response to enforce in case of a sequence failure, or a failure to execute within the timeout period

In the case that the schedule calls for a non-deterministic scan of some segment (i.e. a scan that is only performed with some probability), the WDT MAY already choose random data to determine if the segment scan will be performed on the next run; if not, it SHOULD expect the segment to be scanned on the next run.

For performance and simplicity reasons, it is RECOMMENDED **not** to configure the WDT so that more than one segment is scheduled to be scanned on any execution of the PRMVA, and the WDT MAY reject configuration data that calls for multiple scans on any one execution run.

The maximum allowed period to the next run MUST be set to the minimum of the timeoutPeriod parameters across all segments that are scheduled to be scanned next, and the error response in case of a sequence failure, or a failure to execute MUST be at least as strong as the errorResponse parameter for each segment that is scheduled to be scanned next.

The pre-defined errorResponse values are:

0 = The minimum required mandatory error response, as specified in [9], Section 7.3.1.

1 = The engine is forced to reboot

2-255 = Reserved values

Other values are manufacturer specific.

Response 1 is always considered as being stronger than response 0. The strength ordering of manufacturer-specific values MUST be determined by the manufacturer.

5.5.1 TBB_ConfigureWDTMeasurements

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TBB_ORD_ConfigureWDTMeasurements
4	16	2S	16	BYTE[16]	segmentID	Identifier of segment to be scanned
5	4	3S	4	UINT32	addressSize	Length in bytes of segment start and end addresses
6	<>	4S	<>	BYTE []	startAddress	Start address of segment to be scanned
7	<>	5S	<>	BYTE []	endAddress	End address of segment to be scanned
8	4	6S	4	UINT32	rimRunSize	Length in bytes of expected run-time measurement
9	<>	7S	<>	BYTE []	rimRunData	Expected run-time measurement for this segment
10	4	8S	4	UINT32	scheduleSize	Length in bytes of schedule data
11	<>	9S	<>	BYTE []	scheduleData	Schedule at which to scan this segment
12	4	10S	4	UINT32	errorResponse	What to do if the run-time check fails
13	4	11S	4	UINT32	timeoutPeriod	Watchdog timeout period in microseconds
14	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TBB to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
15	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
16	20			TPM_AUTHDATA	resAuth	OPTIONALLY The authorization session digest for the returned parameters. HMAC key: tbbAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TBB_ORD_ ConfigureWDTMeasurements
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TBB to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	OPTIONALLY The authorization session digest for the returned parameters. HMAC key: tbbAuth

Action

The TBB SHALL perform the following steps:

1. If the TCB has **not** identified to the TBB that the originator is a trusted service, validate the command authorization parameters based on *tbbAuth*. If the validation fails then return TPM_AUTHFAIL.
2. If segmentID is a segment already registered as part of the WDT configuration data, and addressSize, rimRunSize, scheduleSize, errorResponse and timeoutPeriod are all zero, then remove this segment from the WDT configuration data, and return TPM_SUCCESS.
3. Otherwise, check the addressSize, startAddress and endAddress describe a valid address range than can be scanned by the PRMVA. If they don't, then return TPM_BAD_PARAMETER.
4. Check that the scheduleData describe a schedule that the PRMVA is able to perform. If they don't, then return TPM_BAD_PARAMETER.

Start of informative comment:

As a concrete example, the scheduleData could consist of four UINT32 parameters p, q, r, and s. The interpretation is that on the nth execution run of the PRMVA since boot, then whenever n mod r = s, the PRMVA will scan this segment with probability p/q.

End of informative comment.

5. Check that the errorResponse is one that the PRMVA is able to perform. If the errorResponse corresponds to an unknown or unimplemented value, then return TPM_BAD_PARAMETER.
6. Check that the timeoutPeriod is at least the minimum that the WDT supports between calls of the PRMVA. If it is too short, then return TPM_BAD_PARAMETER.
7. Otherwise, register all the above parameters as part of the configuration data of the WDT, to be used after the next successful call of *TBB_SeedASC*. If the segmentID refers to a segment that is already registered then overwrite the existing entry; otherwise add a new entry.
8. If the command was authorized, then construct the output authorization parameters.
9. Return TPM_SUCCESS.

5.5.2 TBB_SeedASC

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TBB_ORD_SeedASC
4	4	2S	4	UINT32	seedFragmentSize	Length in bytes of seedFragment which follows
5	<>	3S	<>	BYTE []	seedFragment	String of (possibly encrypted) bytes to use as part of the seed
6	4	4S	4	UINT32	proposedAlgorithm	Identifier for proposed pseudo-random algorithm
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TBB to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	The continue use flage for the authorization session handle.
10	20			TPM_AUTHDATA	authData	OPTIONALLY the authorization session digest for inputs. HMAC key: tbbAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	Ordinal	Command ordinal: TBB_ORD_SeedASC
4	4	3S	4	UINT32	seedRemainderSize	Length in bytes of remainder of seed which follows
5	<>	4S	<>	BYTE []	seedRemainder	String of (possibly encrypted) bytes to use as rest of the seed
6	4	5S	4	UINT32	usedAlgorithm	Identifier for pseudo-random algorithm that will be used
7	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TBB to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
9	20			TPM_AUTHDATA	resAuth	OPTIONALLY The authorization session digest for the returned parameters. HMAC key: tbbAuth

Action

The TBB SHALL perform the following steps:

1. If the TCB has **not** identified to the TBB that the originator is a trusted service, validate the command authorization parameters based on *tbbAuth*. If the validation fails then return TPM_AUTHFAIL.
2. If TBB_SeedASC has already been called this boot cycle, then return TPM_FAIL.
3. Examine the proposedAlgorithm, and decide if it is supported and acceptable in terms of security. If it is supported and acceptable, then set usedAlgorithm = proposedAlgorithm. Otherwise set usedAlgorithm to one of the TBB's preferred algorithms.
4. Parse the seedFragment, decrypting it if necessary. Check that the decryption (if required) succeeds and that the decrypted fragment length does not exceed the seed-length for the appropriate algorithm; if not, then return TPM_BAD_PARAMETER.
5. Otherwise, generate random bits making up the remainder of the seed and encrypt if necessary. Set seedRemainder and seedRemainderLength accordingly.
6. Read the registered configuration data for the WDT, and transfer it to the WDT locked down for use during the current boot-cycle.
7. If the command was authorized, then construct the output authorization parameters.
8. Return TPM_SUCCESS.

5.5.3 TBB_IncrementASC

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	Ordinal	Command ordinal: TBB_ORD_IncrementASC
4	4	2S	4	UINT8	sequenceNumSize	Length in bytes of sequenceNum which follows
5	<>	3S	<>	BYTE []	sequenceNum	The next number in the pseudo-random sequence
6	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TBB to cover inputs
7	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	The continue use flage for the authorization session handle.
9	20			TPM_AUTHDATA	authData	OPTIONALLY the authorization session digest for inputs. HMAC key: tbbAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TBB_ORD_IncrementASC
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TBB to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	OPTIONALLY The authorization session digest for the returned parameters. HMAC key: tbbAuth

Action

The TBB SHALL perform the following steps:

1. If the TCB has **not** identified to the TBB that the originator is a trusted service, validate the command authorization parameters based on *tbbAuth*. If the validation fails then return TPM_AUTHFAIL.
2. Check if the sequenceNum is the correct next number in the pseudo-random sequence. If it isn't, then return TPM_AUTHFAIL, and initiate the configured errorResponse to an integrity failure for this execution run of the PRMVA.
3. Perform each scheduled segment scan – if any – and check the measurement result against the configured rimRunData. If there is a mismatch, then return TPM_AUTHFAIL, and initiate the configured errorResponse to an integrity failure for this segment.
4. If the command was authorized, then construct the output authorization parameters.
5. Return TPM_SUCCESS.

5.6 Additional (Virtual) Monotonic Counters and NV Storage

Start of informative comment:

The use-cases for these ordinals are to allow the engine to use additional monotonic counters, and Virtual NV Storage, and protect them using counterStorageProtect. The functionality is the same as the corresponding TPM ordinals. However, there is a limitation on number of increments of counterStorageProtect, so the virtual counters and NV spaces need to be protected temporarily by the TBB before being signed for protection into the next boot cycle.

End of informative comment.

Where supported, the following TBB ordinals SHALL function as for the corresponding TPM Ordinals, defined in [3], except that TBB_IncrementCounter SHALL NOT store the countID in TPM_STCLEAR_DATA -> countID and SHALL NOT compare countID against a pre-stored TPM_STCLEAR_DATA -> countID to check for a mis-match:

TBB Ordinal	Corresponding TPM Ordinal	Section of [3] where defined
TBB_CreateCounter	*TPM_CreateCounter*	25.1
TBB_IncrementCounter	*TPM_IncrementCounter*	25.2
TBB_ReadCounter	*TPM_ReadCounter*	25.3
TBB_ReleaseCounter	*TPM_ReleaseCounter*	25.4
TBB_ReleaseCounterOwner	*TPM_ReleaseCounterOwner*	25.5
TBB_NV_DefineSpace	*TPM_NV_DefineSpace*	20.1
TBB_NV_WriteValue	*TPM_NV_WriteValue*	20.2
TBB_NV_WriteValueAuth	*TPM_NV_WriteValueAuth*	20.3
TBB_NV_ReadValue	*TPM_NV_ReadValue*	20.4
TBB_NV_ReadValueAuth	*TPM_NV_ReadValueAuth*	20.5

Where any of the above ordinals are supported, then the values of the virtual Monotonic Counters and NV Storage locations MUST be integrity protected, and they SHOULD be protected using the following signature mechanism:

1. The key used to sign the data SHALL be stored as part of the MTM's storage hierarchy.

2. The signed data SHALL include the value of the underlying MTM monotonic counter counterStorageProtect.

3. The signed data SHALL be updated after each ordinal call that changes the value of any counter or NV storage location, or changes the number of counters or NV storage locations.

4. On each boot cycle, the TBB SHALL verify the integrity of the signed data, and SHALL verify that the embedded value of the counterStorageProtect counter is at least as high as the value stored in the MTM. The TBB MAY re-verify the signed data at subsequent points during the boot cycle (e.g. run-time integrity checks).

5. In the event of an integrity failure, the TBB SHALL either trigger a transition of the engine to a "FAILED" state or at minimum ensure that all attempts to use the above-listed ordinals result in a return code of TPM_FAILEDSELFTEST. If the integrity cannot be recovered by a re-boot, then further recovery mechanisms (e.g. back-up signed data) are out of scope.

6. As the NV Storage locations MAY contain confidential data, encrypted blobs SHALL be recorded as part of the signed data, rather than the contents of the storage locations themselves. The key(s) needed to decrypt the blobs SHALL be stored as part of the MTM's storage hierarchy.

7. The authorization data used to increment counterStorageProtect and to access the relevant signing key, and decryption key(s) if necessary, SHALL be stored in the TBB or sealed to the TBB.

By this mechanism, an indefinite number of additional monotonic counters (and virtual NV Spaces) MAY be created and protected. If the above signature mechanism is not used to create additional counters and spaces, then a mechanism achieving at least as strong integrity protection, confidentiality protection and anti-rollback protection of the virtual counters and NV storage MUST be used instead.

Start of informative comment:

As the signed data is only ever generated and used within the TBB, there is no need to define an explicit structure for this data such as a list or tree. Different implementations may also insert additional information such as date and time of creation, date and time of expiry, boot cycle number, tick counter, nonces etc.

End of informative comment.

Some of these TPM ordinals (e.g. TPM_CreateCounter, TPM_ReleaseCounterOwner) are defined in [3] such that they need or may need owner authorization. Where the corresponding TBB ordinals are implemented, it is RECOMMENDED to authorize the TBB ordinals using *tbbAuth* instead of *ownerAuth* and then use TBB_OSAP, TBB_OIAP or TBB_DSAP to establish an authorization session with the TBB so that the TBB ordinals can be executed: see 5.7 below.

5.6.1 TBB_ProtectVirtualCountersAndStorage

Start of informative comment:

To limit wear-out, counterStorageProtect should only be incremented rarely e.g. at most once per boot cycle. It is possible that in some cycles it cannot be incremented, as a different monotonic counter has been incremented; in that case, the increment should happen at the start of the next boot cycle.

End of informative comment.

To limit wear-out, counterStorageProtect SHOULD only be incremented once per boot cycle. Thus values of the virtual counters (and virtual NV space) SHOULD be stored temporarily in the TBB before protecting them into the next boot cycle with an increment of counterStorageProtect. That protection is achieved by the ordinal *TBB_ProtectVirtualCountersandStorage*. This ordinal SHOULD be called on each power-down event (unless there have been no virtual counter or NV updates this boot cycle), and MAY be called by the system after "critical" updates to counters and virtual NV storage locations, so as to protect against risk of uncontrolled power-down. The execution of *TBB_ProtectVirtualCountersandStorage* MAY fail where it is not possible to increment counterStorageProtect because a different MTM counter has already been incremented in this boot cycle. In that case, it is RECOMMENDED to retain the new signed data set of virtual counter values and NV storage locations, and increment counterStorageProtect at the start of the subsequent boot cycle.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TBB_ORD_ProtectVirtualCountersAndStorage
4	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TBB to cover inputs
5	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	The continue use flage for the authorization session handle.
7	20			TPM_AUTHDATA	authData	OPTIONALLY the authorization session digest for inputs. HMAC key: tbbAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TBB_ORD_ProtectVirtualCountersAndStorage
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TBB to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	OPTIONALLY The authorization session digest for the returned parameters. HMAC key: tbbAuth

Action

The TBB SHALL perform the following steps:

1. If the TCB has **not** identified to the TBB that the originator is a trusted service, validate the command authorization parameters based on *tbbAuth*. If the validation fails then return TPM_AUTHFAIL.
2. Verify the existing signed data set. If the verification fails, then return TPM_FAILEDSELFTEST.
3. Otherwise, construct a “to be signed” data set containing the most recent values of all virtual counters, and encrypted blobs with the most recent (encrypted) values of all virtual NV storage locations, and containing the latest value of counterStorageProtect + 1.
4. Sign the data using the signing key stored in the MTM. If this fails, return TPM_FAIL.
5. Store the new signed data within non-volatile storage, e.g. for checking at subsequent boot cycles.
6. Perform TPM_IncrementCounter with countID MTM_PERMANENT_DATA->counterStorageProtectId. If this fails, then return TPM_BAD_COUNTER.
7. If the command was authorized, then construct the output authorization parameters.
8. Return TPM_SUCCESS.

5.7 Use restricted commands (e.g. MakeIdentity/ActivateIdentity)

Start of informative comment:

These ordinals instruct the TBB to start a TPM authorization session, and use its locally stored authData (verificationAuth) to authorize subsequent owner-authorized commands like TPM_MakeIdentity and TPM_ActivateIdentity. Or to authorize other TPM commands that the owner may have delegated.

Effectively, the TBB acts as a trusted Man In The Middle. The TBB component holding verificationAuth calculates and applies the appropriate authorization HMACs in place of the entity calling the owner-authorized commands.

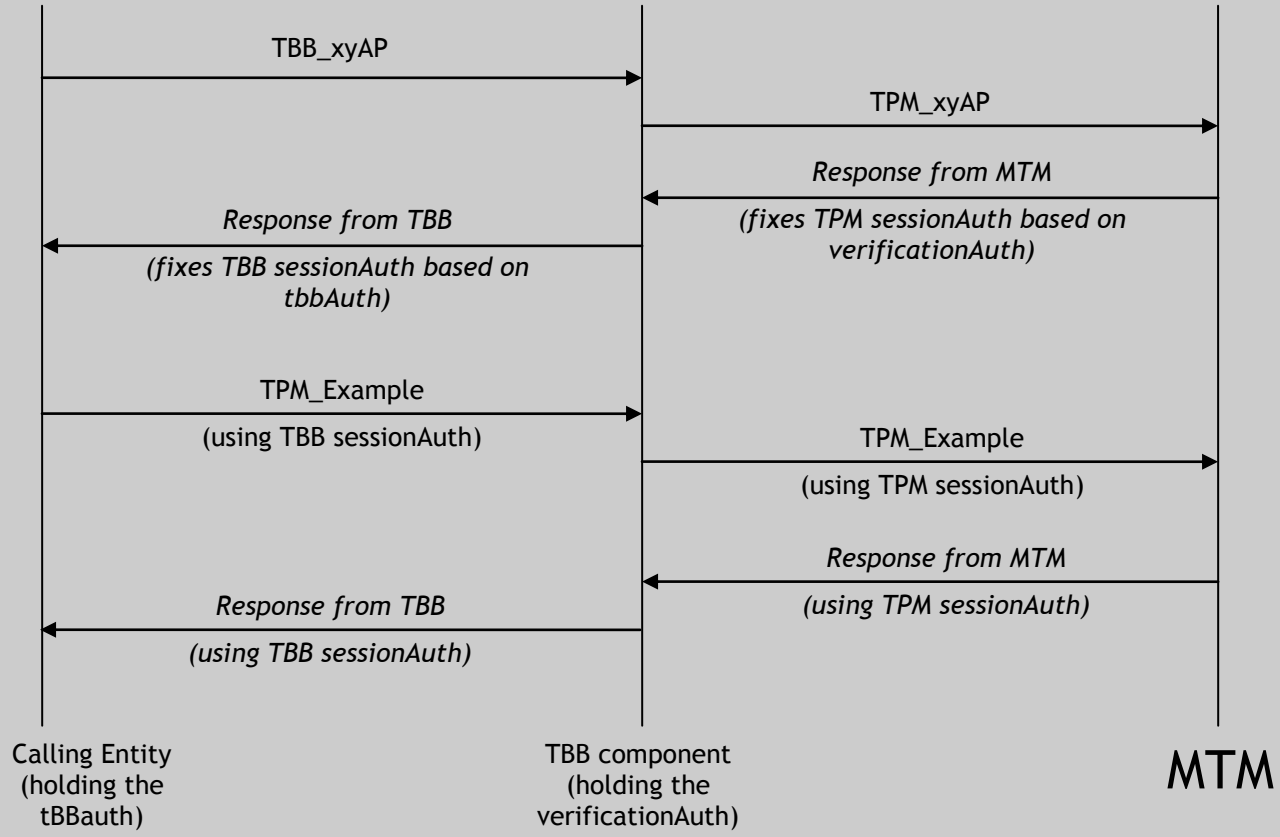


Figure 5. Use of TBB component as a "Trusted Man In The Middle"

An alternative scenario – which is more appropriate for a locally-owned User engine – is that the TBB may at an appropriate point (such as the launch of a trusted application, the first or Nth use of a TDDLI API, etc) prompt the User for the owner authorization data, and then make use of the *ownerAuth* data in order to establish the *sessionAuth* with the MLTM.

End of informative comment.

Where supported, the following TBB ordinals function as for the corresponding TPM Ordinals, defined in [3]:

TBB Ordinal	Corresponding TPM Ordinal	Section of [3] where defined
TBB_OIAP	*TPM_OIAP*	18.1
TBB_OSAP	*TPM_OSAP*	18.2
TBB_DSAP	*TPM_DSAP*	18.3

Where the entity calling owner-authorized commands is implicitly identified to the TBB as privileged, then the calling entity SHOULD use default (all zero) authorization data in place of *tbbAuth*, when initiating the authorization session (and when calculating the HMACs on subsequent commands). Where the calling entity uses its own explicit authorization data, then this data SHALL be *tbbAuth*, and SHALL be used by the calling entity in establishing the session with the TBB (and when calculating the HMACs on subsequent commands).

The MTM is regarded as a sub-component of the TBB. Another component of the TBB (the one which holds the verificationAuth data, or requests the local owner to supply ownerAuth data) SHALL replace the HMACs using the correct verificationAuth or ownerAuth data, and then proxy the session and its enclosed TPM/MTM ordinals to the MTM.

The TBB SHALL have configuration data listing all the owner-authorized TPM and MTM ordinals that the stakeholder allows to be authorized with verificationAuth data. The TBB SHALL also have configuration data listing which of those commands the stakeholder allows to be called by privileged entities outside the TBB, and the TBB SHALL prevent use of the other commands in proxy sessions.

For instance, the configuration data SHOULD prevent such proxy sessions making use of MTM_InstallRIM or from using TPM_IncrementCounter to increment counterRIMProtect or counterStorageProtect. While all these commands can be authorized by verificationAuth, the relevant functionality SHOULD not be directly exposed outside the TBB. Instead, the external entity SHOULD use commands listed in 5.3 and 5.5.

Of course, where the calling entity has its own authorization data, the stakeholder MAY delegate owner-authorized commands directly to that authData. In that case, the calling entity SHOULD use TPM_DSAP.

As discussed under section 5.2 above, a TBB authorization session MAY also be established to call TBB ordinals rather than MTM or TPM ordinals e.g. to call TBB ordinals which can only be accessed from a service which is trusted by the engine stakeholder. In such cases, TBB ordinals MUST be handled directly by the TBB without being relayed on to the MTM.

It is NOT RECOMMENDED to mix TBB and TPM/MTM ordinals within a single authorization session, as the rolling session nonces become hard for the TBB to securely track and manage. The TBB MAY abort an authorization session which mixes ordinals in this way.

6. TBB Interface API

Start of informative comment:

The intent of this document is to describe an interface between the TCG Software Stack (TSS) for an MTM and the Mobile Platform Module (MTM) in an MTM-enabled Trusted Platform. This interface is called the MTM Device Driver Library Interface (MTM DDLI). The MTM device driver library (MTM DDL) is a module that exists between TSS and the low-level MTM device driver (MTM DD). The MTM DDL is implemented in user-mode and performs processing in the calling application context (i.e. TSS core system service). The MTM DDL is designed to be single-threaded, single-instance, and assumes that MTM command serialization has been performed by the calling application. The MTM DDLI is of a synchronous nature. The MTM vendor is responsible for defining the interface between this library and the actual MTM device. The MTM vendor can choose the communication and resource allocation mechanisms between this library and any kernel-mode MTM driver or software MTM simulator.

In most platform implementations, the MTM DLL is loaded when a TSS application (i.e. TCS) initializes. On most platforms, this will occur during operating system startup. To guarantee access to the MTM DDL by any TSS application module, a strict library naming convention must be followed for each operating system implementation.

The "classical" memory allocation approach is used by the TPM DDL and is also followed by the MTM DDL, where the calling application allocates memory for the in and out parameters associated with each interface call. Symmetrically, the calling application is responsible for deallocating the memory associated with any call to the MTM DDL. Retrieving the required parameter size from the callee accepting to call the callee twice or always checking for an error return code "TPMDDL_INSUFFICIENT_BUFFER" is not supported since not every MTM command can be repeated getting the same results (e.g. MTM_VerifyRIMCertAndExtend). In the MTM DDLI described in this document, parameters are documented as follows:

in parameters with the comment "// in"

out parameters with the comment "// out"

End of informative comment.

The following APIs are REQUIRED to be supported by all TDDLI implementations:

Tddli_Open, Tddli_Close, Tddli_Cancel, Tddli_GetCapability, Tddli_SetCapability, Tddli_GetStatus and Tddli_TransmitData.

The following API is RECOMMENDED for all TDDLI implementations:

Tddli_TransmitFilteredData

The following APIs are OPTIONAL for all TDDLI implementations:

Tddli_PowerManagement and Tddli_PowerManagementControl

6.1 TDDL Interface

6.1.1 Tddli_Open

Start of informative comment:

This function establishes a connection with the MTM device driver. Following a successful response to this function, the TPM device driver must be prepared to process TPM command requests from the calling application. The application utilizing the MTM DDL is guaranteed to have exclusive access to the MTM device. If this call fails, it may be an indication that the MTM device driver is not loaded, started, or the MTM cannot support any protected requests. This function must be called before calling Tddli_GetStatus, Tddli_GetCapability, Tddli_SetCapability, or Tddli_TransmitData.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_Open( );
```

Parameters:

None.

Return Value:

```
TDDL_SUCCESS
TDDL_E_COMPONENT_NOT_FOUND
TDDL_E_ALREADY_OPENED
TDDL_E_FAIL
```

6.1.2 Tddli_Close

Start of informative comment:

This function closes a connection with the MTM device driver. Following a successful response to this function, the TPM device driver can clean up any resources used to maintain a connection with the MTM device driver library. If this call fails, it may provide an indication that the MTM device driver cannot clean up or may need to be restarted or reloaded.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_Close( );
```

Parameters:

None.

Return Value:

```
TDDL_SUCCESS
TDDL_E_ALREADY_CLOSED
TDDL_E_FAIL
```

6.1.3 Tddli_Cancel

Start of informative comment:

This function cancels an outstanding MTM command. An application can call this function, in a separate context, to interrupt a MTM command that has not completed. The previous MTM command must be the result of a call to the Tddli_TransmitData function. The MTM device driver must acknowledge this function if it has not returned from a previous TPM command and return TDDL_COMMAND_ABORTED for the call in process.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_Cancel();
```

Parameters:

None.

Return Value:

```
TDDL_SUCCESS
TDDL_COMMAND_COMPLETED
TDDL_E_FAIL
```

6.1.4 Tddli_GetCapability

Start of informative comment:

This function queries the TPM hardware, firmware and device driver attributes such as firmware version, driver version, etc.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_GetCapability
(
    UINT32 CapArea,          // in
    UINT32 SubCap,           // in
    BYTE* pCapBuf,           // out
    UINT32* pCapBufLen       // in, out
);
```

Parameters:

Type	Name	Description
UINT32	CapArea	Partition of capabilities to be interrogated
UINT32	SubCap	Subcode of the requested capabilities.
BYTE*	pCapBuf	Pointer to a buffer containing the received attribute data.
UINT32*	pCapBufLen	[in] Size of the receive buffer in bytes [out] Number of written bytes.

Return Values:

TDDL_SUCCESS
TDDL_E_BAD_PARAMETER
TDDL_E_OUTOFMEMORY
TDDL_E_BADTAG
TDDL_E_FAIL

Defined Capability Areas	**Defined Capability Sub-Codes**	**Response**
TCPA_CAP_VERSION	TSS_CAP_PROP_DRV	Returns the version of the TPM device driver. The version is coded in the TPM_VERSION format.
TCPA_CAP_VERSION	TSS_CAP_PROP_FW	Returns the version of the current TPM firmware. The version is coded in the TPM_VERSION format.
TCPA_CAP_VERSION	TSS_CAP_PROP_FW_DATE	Returns the release date of the firmware. The date is coded in three bytes mm/dd/yy (mm=month, dd=day, yy=year).
TCPA_CAP_PROPERTY	TCPA_CAP_PROP_MANUFACTURER	Returns the name of the device vendor. The returned data is coded in an ASCII string without the trailing null.
TCPA_CAP_PROPERTY	TSS_CAP_PROP_MODULE_TYPE	Returns the vendor specific designation type of the device. The returned data is coded in an ASCII string without the trailing null.
TCPA_CAP_PROPERTY	TSS_CAP_PROP_GLOBAL_STATE	Returns the global state of the module, (e.g. initialized or personalized).
TCPA_CAP_VENDOR	TCPA_CAP_VENDOR_XXX	Returns the vendor specific capabilities of the TPM.

6.1.5 Tddli_SetCapability

Start of informative comment:

This function sets parameters in the TPM hardware, firmware and device driver attributes. An application can set TPM device driver and operating parameters that may be defined by the TPM vendor. For now, the parameter definitions are vendor defined.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_SetCapability
(
    UINT32 CapArea,          // in
    UINT32 SubCap,           // in
    BYTE* pSetCapBuf,        // in
    UINT32 SetCapBufLen      // in
);
```

Parameters:

Type	Name	Description
UINT32	CapArea	Partition of capabilities to be set.
UINT32	SubCap	Subcode of the capabilities to be set.
BYTE*	pCapBuf	Pointer to a buffer containing the capability data to be sent.
UINT32*	pCapBufLen	[in] Size of the request buffer in bytes

Return Values:

```
TDDL_SUCCESS
TDDL_E_OUTOFMEMORY
TDDL_E_BAD_PARAMETER
TDDL_E_BADTAG
TDDL_E_FAIL
```

6.1.6 Tddli_GetStatus

Start of informative comment:

This function queries the status the TPM driver and device. An application can determine the health of the TPM subsystem by utilizing this function.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_GetStatus
(
    UINT32 ReqStatusType,    // in
    UINT32* pStatus,         // out
);
```

Parameters:

Type	Name	Description
UINT32	ReqStatusType	Requested type of status information driver or device
UINT32*	punStatus	[out] Requested status.

Return Values:

TDDL_SUCCESS
TDDL_E_BAD_PARAMETER
TDDL_E_INSUFFICIENT_BUFFER
TDDL_E_FAIL

Defined Status Type	Defined Response Code-Code	Description
TDDL_DRIVER_STATUS	TDDL _DRIVER_OK	TPM driver is functioning okay.
TDDL_DRIVER_STATUS	TDDL _DRIVER_FAILED	TPM driver is not functioning.
TDDL_DRIVER_STATUS	TDDL _DRIVER_NOT_OPENED	Device was found, but the corresponding driver could not be opened.
TDDL_DEVICE_STATUS	TDDL _DEVICE_OK	TPM device is functioning okay.
TDDL_DEVICE_STATUS	TDDL_DEVICE_UNRECOVERABLE	TPM device contains an unrecoverable error.
TDDL_DEVICE_STATUS	TDDL _DEVICE_RECOVERABLE	TPM device contains a recoverable error.
TDDL_DEVICE_STATUS	TDDL _DEVICE_NOT_FOUND	TPM device is not found.

6.1.7 Tddli_TransmitData

Start of informative comment:

The function sends a TPM, MTM or TBB command directly to a TBB device driver, causing the TBB to perform the corresponding operation. This function provides a pass through for the TPM parameter block definitions defined in the TPM 1.2 Main Specification.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_TransmitData
(
    BYTE* pTransmitBuf,           // in
    UINT32 TransmitBufLen,        // in
    BYTE* pRececeiveBuf,          // out
    UINT32* pRececeiveBufLen      // in, out
);
```

Parameters:

Type	Name	Description
BYTE*	pTransmitBuf	Pointer to a buffer containing TPM transmit data.
UINT32	TransmitBufLen	Size of TPM transmit data in bytes.
BYTE*	pRececeiveBuf	Pointer to a buffer containing TPM receive data.
UINT32 *	pRececeiveBufLen	[in] Size of TPM receive buffer in bytes. [out] Number of written bytes.

Return Value:

```
TDDL_SUCCESS
TDDL_E_INSUFFICIENT_BUFFER
TDDL_E_IOERROR
TDDL_E_FAIL
```

6.1.8 Tddli_PowerManagement

Start of informative comment:

Terminology Note: In the following discussion, the term "Higher" indicates a lower Dx value. E.g. D0 is a higher power state than D1. This function sets and queries the TPM's power states. Note: There is no corresponding TSPI interface for this function as power management is considered to be the purview of the system, not individual applications.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_SetPowerManagement
(
    BOOLEAN SendSaveStateCommand,       // in
    UINT32 QuerySetNewTPMPowerState     // in/out
);
```

Parameters:

Type	Name	Description
BOOLEAN	SendSaveStateCommand	TRUE = Instructs driver to send the TPM_SaveState command to the TPM. Other parameters in this function MUST be ignored. Caller MUST set the other "in" parameters to 0. Driver MUST set the "out" parameters to 0. FALSE = Do not send the TPM_SaveState, rather perform actions indicated in the other parameters.

UINT32	QuerySetNewTPMPowerState	**On input, if set to -1,** instructs the driver to return the TPM's current ACPI device state and take no further action. **On input, if not set to -1,** instructs the TPM to enter the ACPI device power state per the value of this parameter as follows: 0 = Request to Enter D0 power state 1 = Request to Enter D1 power state 2 = Request to Enter D2 power state 3 = Request to Enter D3 power state **On output:** Value of TPM's current power management state: 0 = TPM current in D0 power state 1 = TPM current in D1 power state 2 = TPM current in D2 power state 3 = TPM current in D3 power state

Comments:

This function MUST return TDDL_E_FAIL if control of power management is set to the driver using the Tddli_PowerManagementControl or if the Tddli_PowerManagementControl returns or would have returned parameter DriverManagesPowerStates Bit 0 = 0.

If the parameter SendSaveStateCommand == FALSE and the TPM cannot maintain its internal state in the requested power state (i.e., without the TPM_SaveState command being issued), this function MUST fail with return value = TDDL_E_BAD_PARAMETER. If the TPM does not support the requested power state, this function MUST return one of the following two failures:

- TDDL_E_FAIL: Output parameter QuerySetNewTPMPowerState **is not** valid

- TDDL_E_BAD_PARAMETER: Output parameter QuerySetNewTPMPowerState **is** valid

Return Values:

TDDL_SUCCESS
TDDL_E_BAD_PARAMETER
TDDL_E_FAIL

6.1.9 Tddli_PowerManagementControl

Start of informative comment:

This command determines and sets which component, TCS or the Driver, receives and handles the platform's OS power state management signals.

This function should be called only during TSS initialization and is not intended to be used to toggle or change the handler of the platform's OS power state management signals. During TSS installation and subsequently during TSS initialization the TCS should query the driver.

During TSS installation, the state of the returned value may be saved for later TSS initialization without performing the query, thus, making the assumption the driver doesn't change. However if there is a chance the driver may change the handling of the platform's OS power state management signals post TSS installation, upon each TSS initialization, the TCS should query the driver.

Note: There is no corresponding TSPI interface for this function as power management is considered to be the purview of the system, not individual applications.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_SetPowerManagement
(
    UINT32 SetPowerManager, // in
    UINT32 DriverManagesPowerStates // out
);
```

Parameters:

Type	Name	Description
UINT32	SetPowerManager	Queries the driver or sets the driver to expect either the TCS or the driver to handle the TPM's power management signals. NOTE, these are bit maps, not values. **Bit 0 = 0**: No change to handler. Used to query the driver if it can/must handle the platform's OS power management signals. Output parameter DriverManagesPowerStates is valid. All other bits are ignored and MUST be zero. **Bit 0 = 1**: indicate change of platform's OS power management handler per bits 1 – 31. Output parameter DriverManagesPowerStates is not valid and must be ignored. **Bit 1 = 0**: Sets the driver to not handle the platform's OS power management signal (i.e., TPM's power state management will be handle by the TCS using the Tddli_PowerManagement function.) If the driver requires that it must handle the TPM's power state, this function MUST perform no action and return error TDDL_E_BAD_PARAMETER. **Bit 1 = 1**: Sets the driver to handle and

		manage power management signal (i.e., TPM's power state management will not be handle by the TCS using the Tddli_PowerManagement function.) If the driver cannot handle the power managment signals, this function MUST return TDDL_E_BAD_PARAMETER Bit 2 - 31: undefined and MUST be 0.
UINT32	DriverManagesPowerStates	**Bit 0 = 0:** driver will handle all platform OS power state management signals and will return an error for calls to Tddli_PowerManagement. **Bit 0 = 1:** driver expects platform's power management signals to be managed by TCS using the Tddli_PowerManagement function. **Bit 1 – 4:** If bit 0 == 1; is a bit map of the supported ACPI device power states that can be controlled by Tddli_PowerManagement QuerySetNewTPMPowerState function. Bit 1 = D0 Bit 2 = D1 Bit 3 = D2 Bit 4 = D3 Bit 5-31 reserved

Return Values:

TDDL_SUCCESS
TDDL_E_BAD_PARAMETER
TDDL_E_FAIL

6.2 TBB Interface Implementation

6.2.1 Unfiltered TDDLI

Start of informative comment:

Unfiltered TDDLI, or what might be more appropriately called *explicitly* authorized TDDLI, allows a calling application to request use of any unrestricted MTM or TBB ordinal. It also allows use of restricted MTM or TBB ordinals, but only if the TBB accepts such ordinals via this channel, and only after presenting explicit authorization data.

This API is to be used either for those ordinals which require no authorization, or for all ordinals in case the TCB does not support a privilege model and the privileged TDDLI. In the latter case, the TBB will need to decide whether to allow a particular calling application to make use of an ordinal that is restricted to the engine stakeholder. This requires trusted applications to have access to a special tbbAuth authorization data. In such an implementation, there is a design assumption on the TCB that it will ensure that only services trusted by the engine's stakeholder have access to the relevant authorization data.

End of informative comment.

The unfiltered TDDLI uses Tddli_TransmitData to send a MTM ordinal (5.1) or other TBB ordinal (5.2) through to the TBB. There is no filtering of data from the calling application before passing it to the MTM or TBB, because all the authorization decisions are made within the TBB, rather than by the TCB.

The TBB MAY be configured to allow only the unrestricted ordinals to be used through the unfiltered TDDLI i.e. only those ordinals identified with a "1" in column 3 of the ordinal table in Section 5.1.2 or with a "1" in column 2 of the ordinal table in Section 5.2. The TBB SHOULD be so-configured where the engine stakeholder expects the filtered TDDLI to be used for all restricted ordinals. If the TBB is so-configured, then all attempts to use restricted ordinals through the unfiltered TDDLI SHALL be aborted with a return code of TDDL_E_FAIL.

Alternatively, the TBB MAY allow all ordinals to be used through the unfiltered TDDLI. In that case, the calling application MUST use *tbbAuth* data and a TBB Authorization Session, in order to execute the restricted ordinals identified with a "1" in column 3 of the ordinal table in Section 5.2.

The calling application MAY also use *ownerAuth* data (or a delegate to the *ownerAuth* data) and a TPM Authorization Session, in order to execute the restricted ordinals identified with a "1" in column 4 of the ordinal table Section 5.1.2. Alternatively, the calling application MAY again use *tbbAuth* data, and rely on the TBB proxying these restricted ordinals to the MTM, as described in Section 5.7.

There is a design assumption on the TCB that it will ensure only services trusted by the engine's stakeholder have access to the relevant authorization data.

6.2.2 Filtered TDDLI

Start of informative comment:

Filtered TDDLI, or what might be more appropriately called implicitly authorized TDDLI, differs from the standard unfiltered TDDLI in that calling applications don't require an authdata. Instead the TCB will create (or reuse as appropriate) a secure session with the TBB, and prevent untrusted applications from making use of this session.

In such an implementation, there is a design requirement on the TCB to ensure that only applications that are trusted by the engine stakeholder will be allowed to use this API. Thus trusted applications running in the lower-security operating system's standard execution space need not worry about storing authorization data. Since there is an implicit use of authdata, the TCB and/or TBB should apply extra integrity checks (at boot-time and during run-time) upon the calling application to ensure the validity of the application.

End of informative comment.

6.2.2.1 Tddli_TransmitFilteredData

Start of informative comment:

The function sends a TPM, MTM or TBB command directly to a TBB device driver, causing the TBB to perform the corresponding operation. This function provides a pass through for the TPM parameter block definitions are defined in the TPM 1.2 Main Specification.

For ordinals that require authData to operate, it is recommended to establish a TBB authorization session using constant all zero data.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_TransmitFilteredData
(
    BYTE* pTransmitBuf,           // in
    UINT32 TransmitBufLen,        // in
    BYTE* pRececeiveBuf,          // out
    UINT32* pRececeiveBufLen      // in, out
);
```

Parameters:

Type	Name	Description
BYTE*	pTransmitBuf	Pointer to a buffer containing TPM transmit data.
UINT32	TransmitBufLen	Size of TPM transmit data in bytes.
BYTE*	pReceiveBuf	Pointer to a buffer containing TPM receive data.
UINT32 *	pReceiveBufLen	[in] Size of TPM receive buffer in bytes. [out] Number of written bytes.

Return Value:

```
TDDL_SUCCESS
TDDL_E_INSUFFICIENT_BUFFER
TDDL_E_IOERROR
TDDL_E_FAIL
TDDL_E_NOT_AUTHORIZED
TDDL_E_OUTOFMEMORY
```

The TBB SHOULD be configured to support Tddli_TransmitFilteredData. If it is so-configured, then:

1. The TCB and TBB MUST establish a secure channel for use with Tddli_TransmitFilteredData (e.g. a transport session or equivalent). There is a design assumption on the TCB that it will protect the channel from spoofing and data manipulation.
2. There is a design assumption on the TCB that it will verify the calling application is a service trusted by the engine stakeholder. The TCB or TBB SHOULD verify that the calling application has not been tampered with by, for instance, requesting a validation of the caller's RIM_run certificate. If the TBB performs the RIM_run validation, it MAY schedule it once every Nth call per session, although the first call per session MUST always be checked. The TBB SHOULD in any case perform a validation of the TCB's RIM_run certificate before allowing any use of restricted ordinals.

If the authorization checks fail, then the return value SHALL be TDDL_E_NOT_AUTHORIZED.

It is RECOMMENDED that Tddli_TransmitFilteredData is used only for the functions listed within the table in 5.1.2 and identified with a "1" in the third column labeled "Restricted to Engine Stakeholder (or Proxy, inc TBB) and those within the table in 5.2 and identified with a "1" in the third column labeled "Restricted to Engine Stakeholder or Proxy". Other functions appearing in the tables MAY be passed through directly for execution, just as if Tddli_Transmit had been called directly.

Within a single session delimited by Tddli_Open and Tddli_Close Tddli_TransmitFilteredData and Tddli_Transmit MAY be mixed and matched as desired. The TBB SHOULD support as many filtered sessions as simultaneous Tddli_Opens.

6.3 Recommended Interface API

6.3.1 Use of Java in Mobile Platforms

Start of informative comment:

As Java is the de-facto standard for application development for most mobile phones, it is expected that Java will be the preferred development language. However, due to immaturity in Java support for Trusted Computing and a multitude of extensions to the core of the Java language across phone platforms, it is impossible at this stage to recommend any specific implementation.

However, in the future projects such as the Java TSS in the Open TC Project (see http://trustedjava.sf.net) and the JCP (see http://jcp.org/en/jsr/detail?id=321) may provide an architecture in which to implement the Abstraction Layer.

End of informative comment.

7. Trusted Services API

Start of informative comment:

The Mobile Abstraction Layer Specification defines Trusted Services APIs in order to facilitate software design without use of MTM ordinals and to provide a logic for one engine to export trusted services to another.

End of informative comment.

Trusted Services APIs SHOULD provide these functionalities:

- Support for verified extend and simple extend of measurements made in post OS environment
- Capture of run-time measurements and comparison with previous measurements
- Support applications to create and use keys, seal and unseal, read and quote PCRs
- Support calls to secure part of the RIM Conversion Agent to create/modify internal RIM Certificates

In considering them, Trusted Services APIs are almost same to TCSI and TSPI in TSS Specification [10], except specific features of mobile platforms. TSS APIs can cover all of them. So Trusted Services APIs are redefined, basically using TSS Specification.

The definition of Trusted Services APIs MAY be roughly composed of three parts:

1. Classification of TSPI and TCSI
   - TSPI and TCSI are classified by mandatory(M)/optional(O)/excluded(E) for Locally-Owned Engine or Remotely-Owned Engine.
   - APIs for local verification function are added, referring to TSS.Nxt Specification [11]
2. Addition of Trusted Services API related to TBB
3. Definition of an interface between two engines(at TSPI level or higher layer)
   - When the DM Engine provides Trusted Services to other Engines because they don't have their own hardware, an interface between engines SHOULD be defined (interface #1 in Figure 1)
   - In some cases, the DM Engine needs an API in order to provide the dependent engine with its own TSPi

7.1 Implementation at TCSI Level of TSS

Start of informative comment:

In this section, APIs of TCSI level are classified and TBB-related APIs are defined.

End of informative comment.

7.1.1 Classification of TCSI

The Table below shows TCSI APIs, classified by Mandatory(M)/Optional(O)/Excluded(E) for Locally-Owned Engine and Remotely-Owned Engine.

Total Required		47	61
No	TCSI API	Required for MRTM	Required for MLTM
1	Tcsip_ChangeAuth	M	M
2	Tcsip_ChangeAuthOwner	O	M
3	Tcsi_OpenContext	M	M
4	Tcsi_GetCapability	M	M
5	Tcsi_FreeMemory	M	M
6	Tcsi_CloseContext	M	M
7	Tcsi_RegisterKey	M	M
8	Tcsip_UnregisterKey	M	M
9	Tcsi_EnumRegisteredKeys	M	M
10	Tcsi_GetRegisteredKey	M	M
11	Tcsi_GetRegisteredKeyBlob	M	M
12	Tcsip_GetRegisteredKeyByPublicInfo	M	M
13	Tcsip_LoadKeyByBlob	M	M
14	Tcsip_LoadKeyByUUID	M	M
15	Tcsip_KeyControlOwner	O	O
16	Tcsi_GetPcrEvent	O	O
17	Tcsi_GetPcrEventsByPcr	O	O
18	Tcsi_GetPcrEventLog	O	O
19	Tcsip_OIAP	M	M
20	Tcsip_OSAP	M	M
21	Tcsip_DSAP	O	O
22	Tcsip_LoadKeyByBlob	M	M
23	Tcsip_LoadKeyByUUID	M	M
24	Tcsip_LoadKey2ByBlob	M	M
25	Tcsip_GetPubKey	M	M
26	Tcsip_CertifyKey	M	M
27	Tcsip_CertifyKey2	O	O
28	Tcsip_CreateWrapKey	M	M
29	Tcsip_EvictKey	M	M
30	Tcsip_AuthorizeMigrationKey	O	M
31	Tcsip_CreateMigrationBlob	O	M
32	Tcsip_ConvertMigrationBlob	O	M
33	Tcsip_UnBind	M	M
34	Tcsip_Unseal	M	M

35	Tcsip_Seal	M	M
36	Tcsip_Sealx	M	M
37	Tcsip_Sign	M	M
38	Tcsip_MigrateKey	O	E
39	Tcsip_CMK_CreateBlob	O	O
40	Tcsip_CMK_ConvertMigration	O	O
41	Tcsip_SelfTestFull	M	M
42	Tcsip_ContinueSelfTest	E	E
43	Tcsip_GetTestResult	M	M
44	Tcsip_SetOwnerInstall	E	M
45	Tcsip_OwnerSetDisable	E	M
46	Tcsip_PhysicalEnable	E	M
47	Tcsip_PhysicalDisable	E	M
48	Tcsip_PhysicalSetDeactivated	E	M
49	Tcsip_SetTempDeactivated	E	M
50	Tcsip_Set_Operator_Auth	E	O
51	Tcsip_TakeOwnership	O	M
52	Tcsip_OwnerClear	O	M
53	Tcsip_ForceClear	E	M
54	Tcsip_DisableOwnerClear	O	M
55	Tcsip_DisableForceClear	E	M
56	Tcsip_PhysicalPresence	E	O
57	Tcsip_GetCapability	M	M
58	Tcsip_SetCapability	O	O
59	(Tcsi_GetCapability)	O	O
60	Tcsip_GetAuditDigest	O	O
61	Tcsip_GetAuditDigestSigned	O	O
62	Tcsip_SetOrdinalAuditStatus	O	O
63	Tcsip_FieldUpgrade	O	O
64	Tcsip_SetRedirection	O	O
65	Tcsip_CMK_SetRestrictions	O	O
66	Tcsip_CMK_ApproveMA	O	O
67	Tcsip_CMK_CreateKey	O	O
68	Tcsip_CMK_CreateTicket	O	O
69	Tcsip_CreateMaintenanceArchive	O	O
70	Tcsip_LoadMaintenanceArchive	O	O
71	Tcsip_KillMaintenanceArchive	O	O
72	Tcsip_LoadManufaturerlMaintenancePub	O	O
73	Tcsip_ReadManufacturerMaintenancePub	O	O
74	Tcsip_GetRandom	M	M
75	Tcsip_StirRandom	M	M
76	Tcsip_CreateEndorsementKeyPair	O	M
77	Tcsip_CreateRevocableEndorsementKeyPair	O	O
78	Tcsip_RevokeEndorsementKeyPair	O	O
79	Tcsip_ReadPubek	O	M
80	Tcsip_MakeIdentity	O	M
81	Tcsip_ActivateTPMIdentity	O	M
82	Tcsip_Extend	M	M

83	Tcsip_PcrRead	M	M
84	Tcsip_Quote	M	M
85	Tcsi_PcrReset	O	O
86	Tcsip_Quote2	M	M
87	Tcsip_Delegate_Manage	O	O
88	Tcsip_Delegate_CreateKeyDelegation	O	O
89	Tcsip_Delegate_CreateOwnerDelegation	O	O
90	Tcsip_Delegate_LoadOwnerDelegation	O	O
91	Tcsip_Delegate_ReadTable	O	O
92	Tcsip_Delegate_UpdateVerificationCount	O	O
93	Tcsip_Delegate_VerifyDelegation	O	O
94	Tcsip_NV_DefineOrReleaseSpace	O	O
95	Tcsip_NV_WriteValue	O	O
96	Tcsip_NV_WriteValueAuth	O	O
97	Tcsip_NV_ReadValue	O	O
98	Tcsip_NV_ReadValueAuth	O	O
99	Tcsi_TPM_ReadCurrentTicks	O	O
100	Tcsip_TickStampBlob	O	O
101	Tcsip_EstablishTransport	M	M
102	Tcsip_ExecuteTransport	M	M
103	Tcsip_CreateCounter	O	O
104	Tcsip_IncrementCounter	M	O
105	Tcsip_ReadCounter	M	M
106	Tcsip_ReleaseCounter	O	O
107	Tcsip_ReleaseCounterOwner	O	O
108	Tcsip_TPM_DAA_Join	O	O
109	Tcsip_TPM_DAA_Sign	O	O
110	Tcsip_EvictKey	M	M
111	Tcsip_TerminateHandle	M	M
112	Tcsip_DirWriteAuth	E	E
113	Tcsip_DirRead	E	E
114	Tcsip_ChangeAuthAsymStart	E	E
115	Tcsip_ChangeAuthAsymFinish	E	E
116	Tcsip_CertifySelfTest	E	E
117	Tcsip_OwnerReadPubek	O	O
118	Tcsip_DisablePubekRead	O	O
119	Tcsip_GetCapabilityOwner	E	E
120	Tcsip_GetCapabilitySigned	E	E
121	Tcsip_InstallRIM	M	O
122	Tcsip_LoadVerificationKey	M	O
123	Tcsip_MTM_LoadVerificationRootKeyDisable	E	O
124	Tcsip_VerifyRIMCert	M	O
125	Tcsip_VerifyRIMCertAndExtend	M	O
126	Tcsip_IncrementBootstrapCounter	M	O
127	Tcsip_SetVerifiedPCRSelection	E	O

7.1.2 Additional APIs related to TBB

Trusted Services APIs SHOULD provide additional functions which are required for TBB operations. They include APIs for secure boot, RIM Certificates, Watchdog Timer, NV storage & monotonic counters and owner-delegated commands.

No	TBB API	TCSI Level	Comments
1	TBB_AddExternalRIMData	Tcsip_AddExternalRIMData	New
2	TBB_RemoveExternalRIMData	Tcsip_RemoveExternalRIMData	New
3	TBB_InternalizeRIMCertificate	Tcsip_InternalizeRIMCertificate	New
4	TBB_GenerateInternalRIMCertificate	Tcsip_GenerateInternalRIMCertificate	New
5	TBB_RefreshInternalRIMCertificate	Tcsip_RefreshInternalRIMCertificate	New
6	TBB_FindInternalRIMCertificate	Tcsip_FindInternalRIMCertificate	New
7	TBB_EnumInternalRIMCertificates	Tcsip_EnumInternalRIMCertificate	New
8	TBB_RemoveInternalRIMCertificate	Tcsip_RemoveInternalRIMCertificate	New
9	TBB_ConfigureWDTMeasurements	Tcsip_ConfigureWDTMeasurements	New
10	TBB_SeedASC	Tcsip_SeedASC	New
11	TBB_IncrementASC	Tcsip_IncrementASC	New
12	TBB_CreateCounter	Tcsip_CreateCounter	5.8.2.14.2
13	TBB_IncrementCounter	Tcsip_IncrementCounter	5.8.2.14.3
14	TBB_ReadCounter	Tcsip_ReadCounter	5.8.2.14.1
15	TBB_ReleaseCounter	Tcsip_ReleaseCounter	5.8.2.14.4
16	TBB_ReleaseCounterOwner	Tcsip_ReleaseCounterOwner	5.8.2.14.5
17	TBB_NV_DefineSpace	Tcsip_NV_DefineOrReleaseSpace	5.8.2.10.1
18	TBB_NV_WriteValue	Tcsip_NV_WriteValue	5.8.2.10.2
19	TBB_NV_WriteValueAuth	Tcsip_NV_WriteValueAuth	5.8.2.10.4
20	TBB_NV_ReadValue	Tcsip_ReadValue	5.8.2.10.5
21	TBB_NV_ReadValueAuth	Tcsip_ReadValueAuth	5.8.2.10.6
22	TBB_ProtectVirtualCountersAndStorage	Tcsip_ProtectVirtualCountersAndStorage	New
23	TBB_OIAP	Tcsip_OIAP	5.8.2.2.3
24	TBB_OSAP	Tcsip_OSAP	5.8.2.2.4
25	TBB_DSAP	Tcsip_DSAP	5.8.2.9.1

7.1.2.1 Tcsip_AddExternalRIMData

Start of informative comment:

The Tcsip_AddExternalRIMData adds external RIM data(including RIM_Certs, RIM_Auth_Certs and Validity Lists) to an external RIM database.

This API primarily provides a stack-based interface to the underlying TBB command.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_AddExternalRIMData
(
    TCS_CONTEXT_HANDLE hContext,              // in
    UINT32             rimDataTag,            // in
    UINT32             rimDataSize,           // in
    BYTE*              rimData,               // in
    UINT32             extensionSize,         // in
    BYTE*              extensionData,         // in
    UINT32             rimDataHandle,         // out
    UINT32             matchingDataSize,      // out
    BYTE*              matchingDataHandles,   // out
    BOOL               validityStatus,        // out
    UTCTIME            validityLimit,         // out
    TPM_AUTH*          verificationAuth       // in, out
)
```

IDL Definition:

```
[helpstring("method Tcsip_AddExternalRIMData")]
TSS_RESULT Tcsip_AddExternalRIMData
(
    [in] TCS_CONTEXT_HANDLE                  hContext,
    [in] UINT32                              rimDataSize,
    [in, size_is(rimData)] BYTE*             rimData,
    [in] UINT32                              extensionSize,
    [in, size_is(extensionData)] BYTE*       extensionData,
    [out] UINT32                             rimDataHandle,
    [out] UINT32                             matchingDataSize,
    [out, size_is(matchingDataHandles)] BYTE* matchingDataHandles,
    [out] BOOL                               validityStatus,
    [out] UTCTIME                            validityLimit,
    [in, out] TPM_AUTH*                      verificationAuth
)
```

Parameters

Type	Name	Description

TCS_CONTEXT_HANDLE	hContext	Handle of established context
UINT32	rimDataTag	Type of RIM data being added (RIM_Cert, RIM_Auth Cert, Validity List,..)
UINT32	rimDataSize	Size of following RIM data
BYTE*	rimData	The actual RIM data
UINT32	extensionSize	Size of extension data
BYTE*	extensionData	Extension data that is to be stored with the RIM data
UINT32	rimDataHandle	Handle for the stored RIM data
UINT32	matchingDataSize	Size of following RIM handles
BYTE*	matchingDataHandles	Handles for matching RIM data already in database
BOOL	validityStatus	Is the RIM data valid?
UTCTIME	validityLimit	Last date and time at which the RIM data is valid
TPM_AUTH	verificationAuth	Authorization

Remarks

7.1.2.2 Tcsip_RemoveExternalRIMData

Start of informative comment:

The Tcsip_RemoveExternalRIMData removes external RIM_Certs, RIM_Auth_Certs and Validity Lists from a RIM database.

This API primarily provides a stack-based interface to the underlying TBB command.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_RemoveExternalRIMData
(
    TCS_CONTEXT_HANDLE      hContext,           // in
    UINT32                  handle,             // in
    TPM_AUTH*               verificationAuth    // in, out
)
```

IDL Definition:

```
[helpstring("method Tcsip_RemoveExternalRIMData")]
TSS_RESULT Tcsip_RemoveExternalRIMData
(
```

```
    [in] TCS_CONTEXT_HANDLE     hContext,
    [in] UINT32                 handle,
    [in, out] TPM_AUTH*         verificationAuth
)
```

Parameters

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context
UINT32	handle	Handle of external RIM data to remove
TPM_AUTH*	verificationAuth	Authorization

Remarks

7.1.2.3 Tcsip_ InternalizeRIMCertificate

Start of informative comment:

The Tcsip_InternalizeRIMCertificate asks a RIM Conversion Agent to create Internal RIM_Certs from valid external RIM_Certs.

This API primarily provides a stack-based interface to the underlying TBB command.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_InternalizeRIMCertificate
(
    TCS_CONTEXT_HANDLE      hContext,           // in
    UINT32                  handle              // in
    TPM_AUTH*               verificationAuth    // in, out
)
```

IDL Definition:

```
TSS_RESULT Tcsip_InternalizeRIMCertificate
(
    [in] TCS_CONTEXT_HANDLE     hContext,
    [in] UINT32                 handle,
    [in, out] TPM_AUTH*         verificationAuth
```

```
)
```

Parameters

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context
UINT32	handle	Handle of external RIM certificate to internalize
TPM_AUTH*	verificationAuth	Authorization

Remarks

7.1.2.4 Tcsip_GenerateInternalRIMCertificate

Start of informative comment:

The Tcsip_GenerateInternalRIMCertificate asks a RIM Conversion Agent to make a current state measurement and directly create an Internal RIM Cert from that measurement.

This API primarily provides a stack-based interface to the underlying TBB command.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_GenerateInternalRIMCertificate
(
    TCS_CONTEXT_HANDLE      hContext,           // in
    UINT32                  rimTemplateSize,    // in
    TPM_RIM_CERTIFICATE     rimTemplateData,    // in
    UINT32                  extensionSize,      // in
    BYTE*                   extensionData,      // in
    UINT32                  addressSize,        // in
    BYTE*                   startAddress,       // in
    BYTE*                   endAddress,         // in
    UINT32                  addAuthSize,        // in
    BYTE*                   addAuthData,        // in
    UINT32                  handle,             // in
    TPM_AUTH*               verificationAuth    // in, out
)
```

IDL Definition:

```
[helpstring("method Tcsip_GenerateInternalRIMCertificate")]
TSS_RESULT Tcsip_GenerateInternalRIMCertificate
(
    [in] TCS_CONTEXT_HANDLE             hContext,
    [in] UINT32                         rimTemplateSize,
    [in] TPM_RIM_CERTIFICATE            rimTemplateData,
    [in] UINT32                         extensionSize,
    [in, size _is(extensionData)] BYTE* extensionData,
    [in] UINT32                         addressSize,
    [in, size_is(startAddress)] BYTE*   startAddress,
    [in, size_is(endAddress)] BYTE*     endAddress,
    [in] UINT32                         addAuthSize,
    [in, addAuthData] BYTE*             addAuthData,
    [in] UINT32                         handle,
    [in, out] TPM_AUTH*                 verificationAuth
)
```

Parameters

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context
UINT32	rimTemplateSize	Size of following RIM template data
TPM_RIM_CERTIFICATE	rimTemplateData	Template for the internal RIM_Cert to be created
UINT32	extensionSize	Size of extension data
BYTE*	extensionData	Extension data to be bound to the internal RIM_Cert
UINT32	addressSize	Size of segment start and end addresses
BYTE*	startAddress	Start address of segment for reference measurement
BYTE*	endAddress	End address of segment for reference measurement
UINT32	addAuthSize	Size of additional authorization data
BYTE*	addAuthData	Additional authorization data for creating internal RIM_Cert
UINT32	handle	Handle of external RIM data to verify authorization
TPM_AUTH*	verificationAuth	Authorization

Remarks

1
2

7.1.2.5 Tcsip_RefreshInternalRIMCertificate

Start of informative comment:

This module re-signs a valid existing internal RIM Certificate using the current value of counterRIMProtect+1. This API primarily provides a stack-based interface to the underlying TBB command.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_RefreshInternalRIMCertificate
(
    TCS_CONTEXT_HANDLE      hContext,          // in
    RTV_RIMCERTSERIAL       serialNumber,      // in
    TPM_AUTH*               verificationAuth   // in, out
)
```

IDL Definition:

```
[helpstring("method Tcsip_RefreshInternalRIMCertificate")]
TSS_RESULT Tcsip_RefreshInternalRIMCertificate
(
    [in] TCS_CONTEXT_HANDLE         hContext,
    [in] RTV_RIMCERTSERIAL          serialNumber,
    [in, out] TPM_AUTH*             verificationAuth
)
```

Parameters

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context
RTV_RIMCERTSERIAL	serialNumber	Serial number of internal RIM_Cert to refresh
TPM_AUTH*	verificationAuth	Authorization

Remarks

7.1.2.6 Tcsip_RemoveInternalRIMCertificate

Start of informative comment:

This module deletes the RIM_Cert from the RIM database. And it increments counterRIMProtect if the internal RIM Certificate being removed has a counter value >= the current counter.

This API primarily provides a stack-based interface to the underlying TBB command.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_RemoveInternalRIMCertificate
(
    TCS_CONTEXT_HANDLE      hContext,           // in
    RTV_RIMCERTSERIAL       serialNumber,       // in
    TPM_AUTH*               verificationAuth    // in, out
)
```

IDL Definition:

```
[helpstring("method Tcsip_RemoveInternalRIMCertificate")]
TSS_RESULT Tcsip_RemoveInternalRIMCertificate
(
    [in] TCS_CONTEXT_HANDLE         hContext,
    [in] RTV_RIMCERTSERIAL          serialNumber,
    [in, out] TPM_AUTH*             verificationAuth
)
```

Parameters

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context
RTV_RIMCERTSERIAL	serialNumber	Serial number of internal RIM_Cert to be removed
TPM_AUTH*	verificationAuth	Authorization

Remarks

7.1.2.7 Tcsip_FindInternalRIMCertificate

Start of informative comment:

This module retrieves an internal RIM_Cert by serial number.

This API primarily provides a stack-based interface to the underlying TBB command.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_FindInternalRIMCertificate
(
    TCS_CONTEXT_HANDLE      hContext,           // in
    BYTE[8]                 label,              // in
    UINT32                  rimVersion,         // in
    BOOL                    includeExtensions,  // in
    UINT32                  rimCertSize,        // out
    TPM_RIM_CERTIFICATE     rimCertOut,         // out
    UINT32                  extensionSize,      // out
    BYTE*                   extensionData       // out
    TPM_AUTH*               verificationAuth    // in, out
)
```

IDL Definition:

```
[helpstring("module "Tcsip_FindInternalRIMCertificate")]
TSS_RESULT Tcsip_FindInternalRIMCertificate
(
    [in] TCS_CONTEXT_HANDLE             hContext,
    [in] BYTE[8]                        label,
    [in] UINT32                         rimVersion,
    [in] BOOL                           includeExtensions,
    [in] UINT32                         rimCertSize,
    [in] TPM_RIM_CERTIFICATE            rimCertOut,
    [in] UINT32                         extensionSize,
    [in, size_is(extensionData)] BYTE*  extensionData,
    [in, out] TPM_AUTH+                 verificationAuth
)
```

Parameters

Type	Name	Description

TCS_CONTEXT_HANDLE	hContext	Handle of established context
BYTE[8]	label	Label of internal RIM_Cert to retrieve
UINT32	rimVersion	Version of internal RIM_Cert to retrieve in case of ambiguity
BOOL	includeExtensions	Extension data is included?
UINT32	rimCertSize	Size of rimCert data
TPM_RIM_CERTIFICATE	rimCertOut	An internal RIM certificate
UINT32	extensionSize	Size of rimCert extension data
BYTE*	extensionData	Extension data that is stored with the rimCert
TPM_AUTH*	verificationAuth	Authorization

Remarks

7.1.2.8 Tcsip_EnumInternalRIMCertificates

Start of informative comment:

This module returns a list of all serial numbers for internal RIM_Certs.

This API primarily provides a stack-based interface to the underlying TBB command.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_EnumInternalRIMCertificates
(
    TCS_CONTEXT_HANDLE      hContext,          // in
    UINT32                  serialListSize,    // out
    RTV_RIMCERTSERIAL*      serialList,        // out
    TPM_AUTH*               verificationAuth   // in, out
)
```

IDL Definition:

```
[helpstring("module Tcsip_EnumInternalRIMCertificates")]
TSS_RESULT Tcsip_EnumInternalRIMCertificates
(
    [in] TCS_CONTEXT_HANDLE                                          hContext,
    [out] UINT32                                                     serialListSize,
    [out, size_is(serialListSize*RTV_RIMCERTSERIAL)] RTV_RIMCERTSERIAL*  serialList,
```

```
    [in, out] TPM_AUTH*                                    verificationAuth
)
```

Parameters

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context
UINT32	serialListSize	Size in bytes of following list of serial numbers
RTV_RIMCERTSERIAL*	serialList	List of serial numbers of internal RIM_Certs
TPM_AUTH*	VerificationAuth	Authorization

Remarks

7.1.2.9 Tcsip_ConfigureWDTMeasurements

Start of informative comment:

The Tcsip_ConfigureWDTMeasurements configures a Watch-dog Timer.

This API primarily provides a stack-based interface to the underlying TBB command.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_TBB_ConfigureWDTMeasurements
(
    TCS_CONTEXT_HANDLE      *hContext,          // in
    UINT16                  segmentID,          // in
    UINT32                  addressSize,        // in
    BYTE*                   startAddress,       // in
    BYTE*                   endAddress,         // in
    UINT32                  rimRunSize,         // in
    BYTE*                   rimRunData,         // in
    UINT32                  scheduleSize,       // in
    BYTE*                   scheduleData,       // in
    UINT32                  errorResponse,      // in
    UINT32                  timeoutPeriod,      // in
    TPM_AUTH*               verificationAuth    // in, out
)
```

IDL Definition:

```
[helpstring("module Tcsip_ConfigureWDTMeasurements")]
TSS_RESULT Tcsip_ConfigureWDTMeasurements
(
    [in] TCS_CONTEXT_HANDLE             hContext,
    [in] UINT16                         segmentID,
    [in] UINT32                         addressSize,
    [in, size_is(startAddress)] BYTE*   startAddress,
    [in, size_is(endAddress)] BYTE*     endAddress,
    [in] UINT32                         rimRunSize,
    [in, size_is(rimRunData)] BYTE*     rimRunData,
    [in] UINT32                         scheduleSize,
    [in, size_is(scheduleData)] BYTE*   scheduleData,
    [in] UINT32                         errorResponse,
    [in] UINT32                         timeoutPeriod,
    [in, out] TPM_AUTH*                 verificationAuth
)
```

Parameters

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context
BYTE[16]	segmentID	Identifier of segment to be scanned
UINT32	addressSize	Length in bytes of segment start and end addresses
BYTE*	startAddress	Start address of segment to be scanned
BYTE*	endAddress	End address of segment to be scanned
UINT32	rimRunSize	Length in bytes of schedule data
BYTE*	rimRunData	Expected run-time measurement for this segment
UINT32	scheduleSize	Length in bytes of schedule data
BYTE*	scheduleData	Schedule at which to scan this segment
UINT32	errorResponse	What to do if it the rum-time check fails
UINT32	timeoutPeriod	Watchdog timeout period in microseconds
TPM_AUTH*	verificationAuth	Verification

1

2 **Remarks**

3

4

5 ### 7.1.2.10 Tcsip_SeedASC

6 ***Start of informative comment:***

7 Thmodule Tcsip_SeedASC gets a seed value and an identifier for the proposed pseudo-random algorithm for
8 ASC (Algorithm Sequence Checker).

9 This API primarily provides a stack-based interface to the underlying TBB command.

10 ***End of informative comment.***

11 **C-Definition:**

```
12      TSS_RESULT Tcsip_SeedASC
13      (
14          TCS_CONTEXT_HANDLE      hContext,              // in
15          UINT32                  seedFragmentSize,      // in
16          BYTE*                   seedFragment,          // in
17          UINT32                  proposedAlgorithm,     // in
18          UINT32                  seedRemainderSize,     // out
19          BYTE*                   seedRemainder,         // out
20          UINT32                  usedAlgorithm,         // out
21          TPM_AUTH*               verificationAuth       // in, out
22      )
23
```

24 **IDL Definition:**

```
25      [helpstring("module Tcsip_SeedASC")]
26      TSS_RESULT Tcsip_SeedASC
27      (
28          [in] TCS_CONTEXT_HANDLE                 hContext,
29          [in] UINT32                             seedFragmentSize,
30          [in, size_is(seedFragment)] BYTE*       seedFragment,
31          [in] UINT32                             proposedAlgorithm,
32          [out] UINT32                            seedRemainderSize,
33          [out, size_is(seedRemainder)] BYTE*     seedRemainder,
34          [out] UINT32                            usedAlgorithm,
35          [in, out] TPM_AUTH*                     verificationAuth
```

```
)
```

Parameters

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context
UINT32	seedFragmentSize	Length in bytes of seedFragment which follows
BYTE*	seedFragment	String of bytes to use as part of the seed
UINT32	proposedAlgorithm	Identifier for proposed pseudo-random algorithm
UINT32	seedRemainderSize	Length in bytes of remainder of seed which follows
BYTE*	seedRemainder	String of bytes to use as rest of the seed
UINT32	usedAlgorithm	Identifier for pseudo-random algorithm that will be used
TPM_AUTH*	verificationAuth	Authorization

Remarks

7.1.2.11 Tcsip_IncrementASC

Start of informative comment:

This module gets the next pseudo-random number for ASC.

This API primarily provides a stack-based interface to the underlying TBB command.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_IncrementASC
(
    TCS_CONTEXT_HANDLE      hContext,           // in
    UINT8                   sequenceNumSize,    // in
    BYTE*                   sequenceNum,        // in
    TPM_AUTH*               verificationAuth    // in, out
)
```

IDL Definition:

```
[helpstring("module Tcsip_IncrementASC")]
TSS_RESULT Tcsip_IncrementASC
(
```

```
    [in] TCS_CONTEXT_HANDLE          hContext,
    [in] UINT8                       sequenceNumSize,
    [in, size_is(sequenceNum)] BYTE* sequenceNum,
    [in, out] TPM_AUTH*              verificationAuth
)
```

Parameters

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context
UINT8	sequenceNumSize	Length in bytes of sequenceNum which follows
BYTE*	sequenceNum	The next number in the pseudo-random sequence
TPM_AUTH*	verificationAuth	Authorization

Remarks

7.1.2.12 Tcsip_ProtectVirtualCountersAndStorage

Start of informative comment:

This module protects virtual counters and virtual NV storage.

This API primarily provides a stack-based interface to the underlying TBB command.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ProtectVirtualCountersAndStorage
(
    TCS_CONTEXT_HANDLE      hContext,           // in
    TPM_AUTH*               verificationAuth,   // in, out
)
```

IDL Definition:

```
[helpstring("module Tcsip_ProtectVirtualCountersAndStorage")]
TSS_RESULT Tcsip_ProtectVirtualCountersAndStorage
(
    [in] TCS_CONTEXT_HANDLE          hContext,
    [in, out] TPM_AUTH*              verificationAuth,
```

)

Parameters

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context
TPM_AUTH*	verificationAuth	Authorization

Remarks

7.2 Implementation at TSPI Level of TSS

Start of informative comment:

In this section, APIs of TSPI level are classified, TBB-related APIs are added, and interface between two engines is defined.

End of informative comment.

7.2.1 Classification of TSPI

The Table below shows TSPI APIs, classified by Mandatory(M)/Optional(O)/Excluded(E) for Locally-Owned Engine and Remotely-Owned Engine.

Total required		58	64
No	TSPI API	Required for MRTM	Required for MLTM
1	Tspi_SetAttribUint32	M	M
2	Tspi_GetAttribUint32	M	M
3	Tspi_SetAttribData	M	M
4	Tspi_GetAttribData	M	M
5	Tspi_ChangeAuth	M	M
6	Tspi_ChangeAuthAsym	E	E
7	Tspi_GetPolicyObject	M	M
8	Tspi_Context_Create	M	M
9	Tspi_Context_Connect	M	M
10	Tspi_Context_GetCapability	M	M
11	Tspi_Context_FreeMemory	M	M
12	Tspi_Context_Close	M	M
13	Tspi_Context_GetDefaultPolicy	M	M
14	Tspi_Context_CreateObject	M	M
15	Tspi_Context_CloseObject	M	M
16	Tspi_Context_GetTPMObject	M	M
17	Tspi_Context_RegisterKey	M	M
18	Tspi_Context_UnregisterKey	M	M
19	Tspi_Context_GetKeyByUUID	M	M
20	Tspi_Context_GetRegisteredKeysByUUID	M	M
21	Tspi_Context_GetRegisteredKeysByUUID2	M	M
22	Tspi_Context_GetKeyByPublicInfo	M	M
23	Tspi_Context_LoadKeyByBlob	M	M
24	Tspi_Context_LoadKeyByUUID	M	M
25	Tspi_TPM_KeyControlOwner	O	O
26	Tspi_TPM_GetEvent	O	O
27	Tspi_TPM_GetEvents	O	O
28	Tspi_TPM_GetEventLog	O	O
29	Tspi_PcrComposite_SelectPcrIndex	M	M
30	Tspi_PcrComposite_SetPcrValue	M	M
31	Tspi_PcrComposite_GetPcrValue	O	O
32	Tspi_PcrComposite_SetPcrLocality	O	O
33	Tspi_PcrComposite_GetPcrLocality	O	O

34	Tspi_PcrComposite_GetCompositeHash	O	O
35	Tspi_PcrComposite_SelectPcrIndexEx	O	O
36	Tspi_Policy_SetSecret	M	M
37	Tspi_Policy_FlushSecret	O	O
38	Tspi_Policy_AssignToObject	M	M
39	Tspi_Key_LoadKey	M	M
40	Tspi_Key_GetPubKey	M	M
41	Tspi_Key_CertifyKey	M	M
42	Tspi_Key_CertifyKey2	M	M
43	Tspi_Key_CreateKey	M	M
44	Tspi_Key_WrapKey	M	M
45	Tspi_Key_UnloadKey	M	M
46	Tspi_TPM_AuthorizeMigrationTicket	O	M
47	Tspi_Key_CreateMigrationBlob	O	M
48	Tspi_Key_ConvertMigrationBlob	O	M
49	Tspi_Data_Bind	M	M
50	Tspi_Data_Unbind	M	M
51	Tspi_Data_Unseal	M	M
52	Tspi_Data_Seal	M	M
53	Tspi_Data_SealX	M	M
54	Tspi_Hash_Sign	M	M
55	Tspi_Hash_VerifySignature	O	O
56	Tspi_Hash_SetHashValue	O	O
57	Tspi_Hash_GetHashValue	O	O
58	Tspi_Hash_UpdateHashValue	O	O
59	Tspi_Key_MigrateKey	O	E
60	Tspi_Key_CMKCreateBlob	O	O
61	Tspi_Key_CMKConvertMigration	O	O
62	Tspicb_CallbackHMACAuth	O	O
63	Tspicb_CallbackXorEnc	O	O
64	Tspicb_CallbackTakeOwnership	O	O
65	Tspicb_CallbackChangeAuthAsym	O	O
66	Tspicb_CollateIdentity	O	O
67	Tspicb_ActivateIdentity	O	O
68	Tspicb_DAA_Sign	O	O
69	Tspicb_DAA_VerifySignature	O	O
70	Tspi_TPM_SelfTestFull	M	M
71	Tspi_TPM_GetTestResult	M	M
72	Tspi_Set_Operator_Auth	E	O
73	Tspi_TPM_TakeOwnership	O	M
74	Tspi_TPM_ClearOwner	O	M
75	Tspi_TPM_SetStatus	O	M
76	Tspi_TPM_GetStatus	O	M
77	Tspi_TPM_GetCapability	M	M
78	Tspi_TPM_ReturnPlatformClass	M	M
79	Tspi_TPM_GetAuditDigest	O	O
80	Tspi_TPM_SetOrdinalAuditStatus	O	O
81	Tspi_TPM_CMKSetRestrictions	O	O

82	Tspi_TPM_CMKApproveMA	O	O
83	Tspi_TPM_CMKCreateTicket	O	O
84	Tspi_TPM_CreateMaintenanceArchive	O	O
85	Tspi_TPM_LoadMaintenancePubKey	O	O
86	Tspi_TPM_KillMaintenanceFeature	O	O
87	Tspi_TPM_CheckMaintenancePubKey	O	O
88	Tspi_TPM_GetRandom	M	M
89	Tspi_TPM_StirRandom	M	M
90	Tspi_TPM_CreateEndorsementKey	O	M
91	Tspi_TPM_CreateRevocableEndorsementKey	O	O
92	Tspi_TPM_RevokeEndorsementKey	O	O
93	Tspi_TPM_GetPubEndorsementKey	O	M
94	Tspi_TPM_CollateIdentityRequest	O	M
95	Tspi_TPM_ActivateIdentity	O	M
96	Tspi_TPM_PcrRead	M	M
97	Tspi_TPM_PcrExtend	M	M
98	Tspi_TPM_Quote	M	M
99	Tspi_TPM_PcrReset	O	O
100	Tspi_TPM_Quote2	M	M
101	Tspi_TPM_Delegate_AddFamily	O	O
102	Tspi_TPM_Delegate_GetFamily	O	O
103	Tspi_TPM_Delegate_InvalidateFamily	O	O
104	Tspi_TPM_Delegate_CacheOwnerDelegation	O	O
105	Tspi_TPM_Delegate_CreateDelegation	O	O
106	Tspi_TPM_Delegate_ReadTables	O	O
107	Tspi_TPM_Delegate_UpdateVerificationCount	O	O
108	Tspi_TPM_Delegate_VerifyDelegation	O	O
109	Tspi_NV_DefineSpace	O	O
110	Tspi_NV_ReleaseSpace	O	O
111	Tspi_NV_WriteValue	O	O
112	Tspi_NV_ReadValue	O	O
113	Tspi_TPM_ReadCurrentTicks	O	O
114	Tspi_Hash_TickStampBlob	O	O
115	Tspi_Context_SetTransEncryptionKey	M	M
116	Tspi_Context_CloseSignTransport	M	M
117	Tspi_TPM_ReadCounter	M	M
118	Tspi_TPM_DAA_JoinInit	O	O
119	Tspi_TPM_DAA_JoinCreateDaaPubKey	O	O
120	Tspi_TPM_DAA_JoinStoreCredential	O	O
121	Tspi_TPM_DAA_Sign	O	O
122	Tspi_DAA_IssuerKeyVerification	O	O
123	Tspi_DAA_IssueSetup	O	O
124	Tspi_DAA_IssueInit	O	O
125	Tspi_DAA_IssueCredential	O	O
126	Tspi_DAA_VerifyInit	O	O
127	Tspi_DAA_VerifySignature	O	O
128	Tspi_DAA_RevokeSetup	O	O
129	Tspi_DAA_ARDecrypt	O	O

130	Tspi_TPM_DirWrite	E	E
131	Tspi_TPM_DirRead	E	E
132	Tspi_TPM_CertifySelfTest	E	E
133	Tspi_TPM_GetCapabilitySigned	E	E
134	Tspi_MTM_InstallRIM	M	O
135	Tspi_MTM_LoadVerificationKey	M	O
136	Tspi_MTM_LoadVerificationRootKeyDisable	E	O
137	Tspi_MTM_VerifyRIMCert	M	O
138	Tspi_MTM_VerifyRIMCertAndExtend	M	O
139	Tspi_MTM_IncrementBootstrapCounter	M	O
140	Tspi_MTM_SetVerifiedPCRSelection	E	O

7.2.2 Additional APIs at TSPI Level

No	TBB API	TSPI Level	Comments
1	TBB_AddExternalRIMData	Tspi_AddExternalRIMData	New
2	TBB_RemoveExternalRIMData	Tspi_RemoveExternalRIMData	New
3	TBB_InternalizeExternalRIMCertificate	Tspi_InternalizeExternalRIMCertificate	New
4	TBB_GenerateInternalRIMCertificate	Tspi_GenerateInternalRIMCertificate	New
5	TBB_RefreshInternalRIMCertificate	Tspi_RefreshInternalRIMCertificate	New
6	TBB_FindInternalRIMCertificate	Tspi_FindInternalRIMCertificate	New
7	TBB_EnumInternalRIMCertificates	Tspi_EnumInternalRIMCertificates	New
8	TBB_RemoveInternalRIMCertificate	Tspi_RemoveInternalRIMCertificate	New
9	TBB_ConfigureWDTMeasurements	Tspi_ConfigureWDTMeasurements	New
10	TBB_SeedASC	Tspi_SeedASC	New
11	TBB_IncrementASC	Tspi_IncrementASC	New
12	TBB_CreateCounter	Tspi_ReadCurrentCounter	4.3.4.23
13	TBB_IncrementCounter	-	-
14	TBB_ReadCounter	-	-
15	TBB_ReleaseCounter	-	-
16	TBB_ReleaseCounterOwner	-	-
17	TBB_NV_DefineSpace	Tspi_NV_DefineSpace, Tspi_NV_ReleaseSpace	4.3.4.26.5, 4.3.4.26.6
18	TBB_NV_WriteValue	Tspi_NV_WriteValue	4.3.4.26.7
19	TBB_NV_WriteValueAuth	-	-
20	TBB_NV_ReadValue	Tspi_NV_ReadValue	4.3.4.26.8
21	TBB_NV_ReadValueAuth	-	-
22	TBB_ProtectVirtualCountersAndStorage	Tspi_ProtectVirtualConutersAndStorage	New
23	TBB_OIAP	-	-
24	TBB_OSAP	-	-
25	TBB_DSAP	-	-

7.2.2.1 Tspi_AddExternalRIMData

Start of informative comment:

The Tspi_AddExternalRIMData adds external RIM data (including RIM_Certs, RIM_Auth_Certs and Validity Lists) to an external RIM database.

This API primarily provides a stack-based interface to the underlying TBB command.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_AddExternalRIMData
(
    TSS_HTPM        hTPM,              // in
    UINT32          rimDataTag,        // in
    UINT32          rimDataSize,       // in
    BYTE*           rimData,           // in
    UINT32          extensionSize,     // in
    BYTE*           extensionData,     // in
    UINT32          rimDataHandle,     // out
    UINT32          matchingDataSize,  // out
    BYTE*           matchingDataHandles, // out
    BOOL            validityStatus,    // out
    UTCTIME         validityLimit,     // out
)
```

Parameters

Type	Name	Description
TSS_HTPM	hTPM	Handle of the TPM object
UINT32	rimDataTag	Type of RIM data being added(RIM_Cert, RIM_Auth Cert, Validity List,..)
UINT32	rimDataSize	Size of following RIM data
BYTE*	rimData	The actual RIM data
UINT32	extensionSize	Size of extension data
BYTE*	extensionData	Extension data that is to be stored with the RIM data
UINT32	rimDataHandle	Handle for the stored RIM data
UINT32	matchingDataSize	Size of following RIM handles

BYTE*	matchingDataHandles	Handles for matching RIM data already in database
BOOL	validityStatus	Is the RIM data valid?
UTCTIME	validityLimit	Last date and time at which the RIM data is valid

Return values

TSS_SUCCESS

TSS_E_INVALID_HANDLE

TSS_E_BAD_PARAMETER

TSS_E_INTERNAL_ERROR

Remarks

7.2.2.2 Tspi_RemoveExternalRIMData

Start of informative comment:

The Tspi_RemoveExternalRIMData removes external RIM_Certs, RIM_Auth_Certs and Validity Lists from a RIM database.

This API primarily provides a stack-based interface to the underlying TBB command.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_RemoveExternalRIMData
(
    TSS_HTPM        hTPM,           // in
    UINT32          handle,         // in
    TPM_AUTH*       verificationAuth // in, out
)
```

Parameters

Type	Name	Description
TSS_HTPM	hTPM	Handle of established context
UINT32	handle	Handle of external RIM data to remove
TPM_AUTH*	verificationAuth	Authorization

Return values

TSS_SUCCESS

TSS_E_INVALID_HANDLE

TSS_E_BAD_PARAMETER

TSS_E_INTERNAL_ERROR

Remarks

7.2.2.3 Tspi_InternalizeRIMCertificate

Start of informative comment:

The Tspi_InternalizeRIMCertificate asks a RIM Conversion Agent to create Internal RIM_Certs from valid external RIM_Certs.

This API primarily provides a stack-based interface to the underlying TBB command.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_InternalizeRIMCertificate
(
    TSS_HTPM        hTPM,       // in
    UINT32          handle      // in
)
```

Parameters

Type	Name	Description
TSS_HTPM	hTPM	Handle of the TPM object
UINT32	handle	Handle of external RIM certificate to internalize

Return values

TSS_SUCCESS

TSS_E_INVALID_HANDLE

TSS_E_BAD_PARAMETER

TSS_E_INTERNAL_ERROR

Remarks

7.2.2.4 Tspi_GenerateInternalRIMCertificate

Start of informative comment:

The Tspi_GenerateInternalRIMCertificate asks a RIM Conversion Agent to make a current state measurement and directly create an Internal RIM Cert from that measurement.

This API primarily provides a stack-based interface to the underlying TBB command.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_GenerateInternalRIMCertificate
(
    TSS_HTPM              hTPM,              // in
    UINT32                rimTemplateSize,   // in
    TPM_RIM_CERTIFICATE   rimTemplateData,   // in
    UINT32                extensionSize,     // in
    BYTE*                 extensionData,     // in
    UINT32                addressSize,       // in
    BYTE*                 startAddress,      // in
    BYTE*                 endAddress,        // in
    UINT32                addAuthSize,       // in
    BYTE*                 addAuthData,       // in
    UINT32                handle,            // in
)
```

Parameters

Type	Name	Description
TSS_HTPM	hTPM	Handle of the TPM object
UINT32	rimTemplateSize	Size of following RIM template data
TPM_RIM_CERTIFICATE	rimTemplateData	Template for the internal RIM_Cert to be created
UINT32	extensionSize	Size of extension data
BYTE*	extensionData	Extension data to be bound to the internal RIM_Cert
UINT32	addressSize	Size of segment start and end addresses
BYTE*	startAddress	Start address of segment for reference measurement

BYTE*	endAddress	End address of segment for reference measurement
UINT32	addAuthSize	Size of additional authorization data
BYTE*	addAuthData	Additional authorization data for creating internal RIM_Cert
UINT32	handle	Handle of external RIM data to verify authorization

Return values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

7.2.2.5 Tspi_RefreshInternalRIMCertificate

Start of informative comment:
This module re-signs a valid existing internal RIM Certificate using the current value of counterRIMProtect+1.
This API primarily provides a stack-based interface to the underlying TBB command.
End of informative comment.

Definition:

```
TSS_RESULT Tspi_RefreshInternalRIMCertificate
(
    TSS_HTPM            hTPM,           // in
    RTV_RIMCERTSERIAL   serialNumber    // in
)
```

Parameters

Type	Name	Description
TSS_HTPM	hTPM	Handle of the TPM object
RTV_RIMCERTSERIAL	serialNumber	Serial number of internal RIM_Cert to refresh

Return values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

7.2.2.6 Tspi_RemoveInternalRIMCertificate

Start of informative comment:

This module deletes the RIM_Cert from the RIM database. And it increments counterRIMProtect if the internal RIM Certificate being removed has a counter value >= the current counter.

This API primarily provides a stack-based interface to the underlying TBB command.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_RemoveInternalRIMCertificate
(
    TSS_HTPM            hTPM,           // in
    RTV_RIMCERTSERIAL   serialNumber,   // in
)
```

Parameters

Type	Name	Description
TSS_HTPM	hTPM	Handle of the TPM object
RTV_RIMCERTSERIAL	serialNumber	Serial number of internal RIM_Cert to be removed

Return values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

1
2

7.2.2.7 Tspi_FindInternalRIMCertificate

Start of informative comment:

The Tspi_FindInternalRIMCertificate retrieves an internal RIM_Cert by serial number.

This API primarily provides a stack-based interface to the underlying TBB command.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_FindInternalRIMCertificate
(
    TSS_HTPM                hTPM,               // in
    BYTE[8]                 label,              // in
    UINT32                  rimVersion,         // in
    BOOL                    includeExtensions,  // in
    UINT32                  rimCertSize,        // out
    TPM_RIM_CERTIFICATE     rimCertOut,         // out
    UINT32                  extensionSize,      // out
    BYTE*                   extensionData       // out
)
```

Parameters

Type	Name	Description
TSS_HTPM	hTPM	Handle of the TPM object
BYTE[8]	label	Label of internal RIM_Cert to retrieve
UINT32	rimVersion	Version of internal RIM_Cert to retrieve in case of ambiguity
BOOL	includeExtensions	Extension data is included?
UINT32	rimCertSize	Size of rimCert data
TPM_RIM_CERTIFICATE	rimCertOut	An internal RIM certificate
UINT32	extensionSize	Size of rimCert extension data
BYTE*	extensionData	Extension data that is stored with the rimCert

Return values

TSS_SUCCESS

TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

7.2.2.8 Tspi_EnumInternalRIMCertificates

Start of informative comment:
The Tspi_EnumInternalRIMCertificate returns a list of all serial numbers for internal RIM_Certs.
This API primarily provides a stack-based interface to the underlying TBB command.
End of informative comment.

Definition:

```
TSS_RESULT Tspi_EnumInternalRIMCertificates
(
    TSS_HTPM                hTPM,           // in
    UINT32                  serialListSize, // out
    RTV_RIMCERTSERIAL*      serialList,     // out
)
```

Parameters

Type	Name	Description
TSS_HTPM	hTPM	Handle of the TPM object
UINT32	serialListSize	Size in bytes of following list of serial numbers
RTV_RIMCERTSERIAL*	serialList	List of serial numbers of internal RIM_Certs

Return values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

1

2 **Remarks**

3

4 ### 7.2.2.9 Tspi_ConfigureWDTMeasurements

5 ***Start of informative comment:***

6 The Tspi_ConfigureWDTMeasurements configures a Watch-dog Timer.

7 This API primarily provides a stack-based interface to the underlying TBB command.

8 ***End of informative comment***.

9 **Definition:**

```
10  TSS_RESULT Tspi_ConfigureWDTMeasurements
11  (
12      TSS_HTPM        hTPM,           // in
13      UINT16          segmentID,      // in
14      UINT32          addressSize,    // in
15      BYTE*           startAddress,   // in
16      BYTE*           endAddress,     // in
17      UINT32          rimRunSize,     // in
18      BYTE*           rimRunData,     // in
19      UINT32          scheduleSize,   // in
20      BYTE*           scheduleData,   // in
21      UINT32          errorResponse,  // in
22      UINT32          timeoutPeriod,  // in
23  )
24
```

25 **Parameters**

26

Type	Name	Description
TSS_HTPM	hTPM	Handle of the TPM object
BYTE[16]	segmentID	Identifier of segment to be scanned
UINT32	addressSize	Length in bytes of segment start and end addresses
BYTE*	startAddress	Start address of segment to be scanned
BYTE*	endAddress	End address of segment to be scanned
UINT32	rimRunSize	Length in bytes of schedule data
BYTE*	rimRunData	Expected run-time measurement for this segment

UINT32	scheduleSize	Length in bytes of schedule data
BYTE*	scheduleData	Schedule at which to scan this segment
UINT32	errorResponse	What to do if it the rum-time check fails
UINT32	timeoutPeriod	Watchdog timeout period in microseconds

Return values

TSS_SUCCESS

TSS_E_INVALID_HANDLE

TSS_E_BAD_PARAMETER

TSS_E_INTERNAL_ERROR

Remarks

7.2.2.10 Tspi_SeedASC

Start of informative comment:

The Tspi_SeedASC gets a seed value and an identifier for the proposed pseudo-random algorithm for ASC(Algorithm Sequence Checker).

This API primarily provides a stack-based interface to the underlying TBB command.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_SeedASC
(
    TSS_HTPM        hTPM,               // in
    UINT32          seedFragmentSize,   // in
    BYTE*           seedFragment,       // in
    UINT32          proposedAlgorithm,  // in
    UINT32          seedRemainderSize,  // out
    BYTE*           seedRemainder,      // out
    UINT32          usedAlgorithm,      // out
)
```

Parameters

Type	Name	Description
TSS_HTPM	hTPM	Handle of the TPM object
UINT32	seedFragmentSize	Length in bytes of seedFragment which follows
BYTE*	seedFragment	String of bytes to use as part of the seed
UINT32	proposedAlgorithm	Identifier for proposed pseudo-random algorithm
UINT32	seedRemainderSize	Length in bytes of remainder of seed which follows
BYTE*	seedRemainder	String of bytes to use as rest of the seed
UINT32	usedAlgorithm	Identifier for pseudo-random algorithm that will be used

Return values

TSS_SUCCESS

TSS_E_INVALID_HANDLE

TSS_E_BAD_PARAMETER

TSS_E_INTERNAL_ERROR

Remarks

7.2.2.11 Tspi_IncrementASC

Start of informative comment:

The Tspi_IncrementASC gets the next pseudo-random number for ASC.

This API primarily provides a stack-based interface to the underlying TBB command.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_IncrementASC
(
    TSS_HTPM        hTPM,           // in
    UINT8           sequenceNumSize,    // in
    BYTE*           sequenceNum,        // in
)
```

Parameters

Type	Name	Description
TSS_HTPM	hTPM	Handle of the TPM object
UINT8	sequenceNumSize	Length in bytes of sequenceNum which follows
BYTE*	sequenceNum	The next number in the pseudo-random sequence

Return values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

7.2.2.12 Tspi_ProtectVirtualCountersAndStorage

Start of informative comment:
The Tspi_ProtectVirtualCountersAndStorage protects virtual counters and virtual NV storage.
This API primarily provides a stack-based interface to the underlying TBB command.
End of informative comment.

Definition:

```
TSS_RESULT Tspi_ProtectVirtualCountersAndStorage
(
    TSS_HTPM            hTPM            // in
)
```

Parameters

Type	Name	Description
TSS_HTPM	hTPM	Handle of the TPM object

Return values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER

TSS_E_INTERNAL_ERROR

Remarks

7.2.3 Interface Between Engines

Several stakeholders may be involved in a mobile device, and each stakeholder may have its own engine. As stated in Section 2.1, an Engine is defined as a dedicated processor or run-time environment with access to trusted resources that is used to run trusted services and normal services. To consider costs and performance, one physical MTM can be used as a common resource. In this case, other Engines have an interface to DM's engine for access to the MTM.

Start of informative comment:

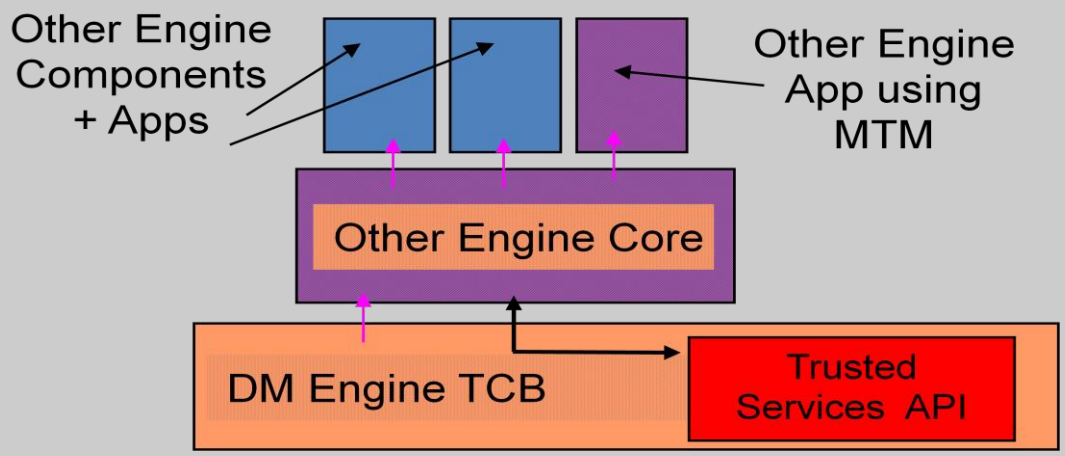


Figure 5. Export of Trusted Services API from DM's Engine; black arrows are API calls, purple arrows are measurements and verifications.

1. Other Engines are not required to have their own hardware, although the SIM card may be regarded as part of the Communication Carriers Engine. Other engines MAY be implemented entirely within the hardware (SoC/Volatile Storage/NV Storage) of the Device Manufacturer's Engine.

2. Where other Engines do not have their own hardware, each such Engine must rely on the TBB of the Device Manufacturer's Engine to provide the engine's own TBB (including its own RTS/RTR).

3. Where other Engines do not have their own hardware, each such Engine should rely on the TCB of the DM Engine to provide control of privileges and appropriate isolation for its Normal Services (e.g. isolation from Services of other Engines).

4. The DM Engine should export a copy of its Trusted Services API to dependent Engines. This enables each dependent Engine to provide a Trusted Services API to its own Normal Services (Refer to Figure 5 above.)

The above requirements will be met through a design that will be compliant with the requirements of the Virtualized Platform Working Group in the case of a solution based upon virtualization. For a non-virtualized solution, the design should at least follow the general outline of the VPWG requirements, thus ensuring that the user of such an interface need not be aware of the particular implementation, thus allowing the implementor to change from a non-virtualized to a virtualized solution without affecting client applications.

Furthermore the design should also compatible with the requirements for TPM.Next.

End of informative comment

During boot-time, the DM Engine's TCB SHOULD measure and verifies the "core" of the dependent engine. For example (the hypervisor model) the TCB MAY measure and verifies the dependent engine's OS kernel or MAY measure and verify its whole OS.

The dependent engine's "core" then SHOULD act as an MVA, and measure and verify other components/apps of the dependent engine, using the Trusted Services API of the DM Engine to record measurements in the engine's MTM.

During run-time, the dependent Engine SHOULD provide to its Normal Services use of the Trusted Services API in the same way as for DM Engine. However, calls to MTM functionality through the Trusted Services API MUST refer to the dependent Engine's MTM rather than the DM's MTM.

[End of document]